NBS-GCR-81-344

Documentation for CFC V, the Fifth Harvard Computer Fire Code

October 1981

Sponsored by
U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Fire Research
Washington, DC 20234

NBS-GCR-81-344

DOCUMENTATION FOR CFC V, THE FIFTH HARVARD COMPUTER FIRE CODE

H. E. Mitler and H. W. Emmons

Harvard University
Division of Applied Sciences
Cambridge, MA

October 1981

NBS Grant G-7-9011, Am. 10

Sponsored by:
U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Fire Research
Washington, DC 20234

Notice

This report was prepared for the Center for Fire Research of the National Engineering Laboratory, National Bureau of Standards, under Grant No. G7-9011, Am. 10.

DOCUMENTATION FOR CFC V, THE FIFTH HARVARD COMPUTER FIRE CODE

Henri E. Mitler and
Howard W. Emmons

Division of Applied Sciences
Harvard University

October 1981

Home Fire Project Technical Report No. 45

TABLE OF CONTENTS

ABSTRACT

This is a complete documentation of the fifth version of the Computer Fire Code (Mark 5). Mark 5 is of course a substantial improvement over Mark 4, which in turn had expanded and generalized Mark 3, etc. Although imposingly thick, this document has been written in a form which may be easily read, as each section begins with a brief outline of that section. The details are then given in the following paragraphs, for the interested reader.

The Computer Fire Code permits the calculation of the evolution of a fire in an enclosure with a number of vents, and containing a number of objects (flammable or otherwise). The fire may be of several kinds, and the calculation will proceed for whatever time the user selects.

Suggestions for the improvement of the program itself or of this document, will be gratefully received. The tape containing the program itself is available at cost. The listing is also available on request (Appendix D).

I. INTRODUCTION

If we watch the development of a fire in an enclosure, we see at first some small ignition process on an individual fuel element. It might be a lighted cigarette on an overstuffed chair, or the spark from a frayed cord under a rug. Eventually, the smoking fuel begins to smolder and then breaks into flames. The flames feed heat back to the fuel, which spreads the fire; it draws air in to produce a hot rising plume; at the ceiling the plume spreads out as a hot gas layer and thus heats the ceiling and by radiation further increases the fire and heats other fuels. Assuming there is enough initial fuel, the hot layer soon deepens and reaches the top of an open door or window and hot gas begins to flow out. Soon, other fuel items in the room are heated to their ignition temperature and the fire spreads rapidly to flashover.

Shortly after the fire starts, value destruction by the deposition on delicate finishes also begins. As soon as the fire starts, the expansion of the air begins to move air throughout the building. Once hot gas begins to flow through a door into another room, the value destruction process is also extended. Furthermore, the toxic gases begin to permeate the building, endangering the lives of the occupants.

Every one of the processes identified above is capable of being individually understood in quantitative detail, either from our present knowledge of physics and chemistry, or (at least) based upon empirical measurements. However, they are all coupled, some of them in quite complex ways. Indeed, the development of a fire even in a simple enclosure--not to speak of one in a large building--involves so many complex interacting phenomena that the calculation of the process before the advent of large-scale high-speed computing machines would have been impossible. Moreover, much of the physics and chemistry has been unknown. Of course, our ignorance and the very complexity of these interacting phenomena is also the source of the inadequacies of some of our current fire safety procedures and standards, which are now based mainly upon experience and committee consensus.

The basic fire research of the past 15 years has begun to fill some of the gaps in our knowledge of flame structure, fire spread rate, soot production, etc., and current research continues to reduce our ignorance. Our present knowledge already suffices to make a good first cut at a theoretical understanding of fire growth in an enclosure. Moreover, the present capacity of generally available computing machines makes it possible to quantitatively predict the fire growth which would occur in a single room, and (probably) even in a multi-room dwelling. Furthermore, the growing capacity and availability of large-scale computing facilities makes numerical fire prediction a practical method for improving public fire safety by permitting the "testing" of any proposed building design, room furniture configuration, etc.

Mathematical modeling of fires is not new: Kawagoe's modeling of the post-flashover fire is over 20 years old.(1) Brief accounts of the history of modeling (mathematical and otherwise) can be found in Babrauskas and Williamson,(2) in Mitler,(3) and in Emmons.(4) Currently, there are at least seven groups of workers in the field of mathematical modeling of fires, including steady-state models, solutions of the Navier-Stokes equations for the flows, stochastic models, etc. [See references (5)-(13)].

As pointed out above, the dynamics of fires are still imperfectly understood and much new research is needed to provide data essential for a fully practical fire prediction. Furthermore, the numerical handling of the equations is still in development and a number of available procedures have yet to be tried.

In this report, we present the progress that has been made in developing a deterministic mathematical model which computes the evolution of a fire in a single room, from ignition through flameover to extinguishment or burnout. This enclosure is assumed to be open to the atmosphere, through at least one rectangular vent in the (vertical) walls. This enclosure may contain (at least in principle) any number of flammable objects.

The Computer Fire Code has been in development for several years. The present version (hereafter referred to as CFC V or Mark V) is a significant improvement over the previous ones in a number of ways which will be described in this report. It has been suggested that these reports be published in looseleaf form, so that as new material emerges, new pages may be added to or replace "old" ones. We may well do this in the future; this version, at any rate, is self-contained. Therefore, there is some overlap with CFC III (Technical Report No. 25). On the other hand, there is little or no overlap with Technical Report No. 34, "The Physical Basis for the Harvard Computer Fire Code" (hereafter referred to as T.R. 34), to which I refer a number of times, especially in Section 5. The whole program has been developed in a form that can be generalized by many different people. To accomplish this Fortran IV is used, since it is a commonly available language.

It may be preaching to the converted, to list here the desirability of having such a mathematical model, but let me briefly outline the advantages, nevertheless:

1. Given the detailed design of an enclosure and its contents, the course of a fire can be followed in detail. This is an incomparably faster and cheaper way to "test" a structure or configuration, than setting up a full-scale model. This is so even though some special small-scale tests may sometimes have to be carried out in order to determine the thermal and related characteristics of special fabrics or other flammable materials.

2. The model can be used instead of many large-scale tests to determine the

relative merits of various materials used in various ways. Thus, to some degree full-scale (corner or other) tests can be postponed to one final decision.

3. As a corollary, we can quickly identify which are the most important parameters determining the course of any particular fire. Hence, details of design could be optimized according to any desired criterion.

4. A further corollary is that performance fire codes would be possible. Thus, a very strong code might say, for example, that "a building will be acceptable from a fire safety point of view only if all occupants will be able to escape no matter where, how, or when a fire starts," according to then best available mathematical model.

5. The deterministic models will interact cooperatively with the stochastic models concurrently being developed.(12) The former models might supply some of the _a priori_ probabilities needed by the stochastic models in _lieu_ of hundreds of test fires. (It is very unlikely that _all_ the needed probabilities can be supplied this way: evidently those representing the behavior of people must be supplied from empirical sources, for the forseeable future).

6. The attempt to build a deterministic model has already revealed a number of gaps in our knowledge which has led to better directions in research programs.

Thus, it is clear that the development of a good mathematical model will become a valuable tool for the architect, builder, city planner, fire code official, researcher, etc. It is impossible to predict how much reduction in fire incidence and severity will result from this effort. However, since fires in the United States result in billions of dollars of damage and many thousands of deaths every year,(14) even a 10% reduction in numbers and severity of fires will be an extraordinary return on investment of money and effort.

I have tried to write this report in a from which is easily readable, by having each section start with a paragraph or two which abstract(s) the contents of the whole section. Thus, the entire paper can be grasped in one relatively brief reading. For the reader interested in more detail, of course, that resides in the remainder.

In section II, a discussion is given of the physics involved. Section III discusses the numerical problems and approaches used by us. Sections IV and V describe the program in considerable detail: V is, in a sense, the heart of the program--the detailed physics is presented there. Section IV discusses most of the rest of the details, especially the structure and logical flow of the program. Sections VI-VIII discuss the use, adequacy, reliability and future of the Computer Fire Code. The listing of the program itself is quite extensive, and is reserved for a separate Appendix, available on request.

II. THE PHYSICS OF THE PROBLEM

In this section we shall briefly discuss the various elemental processes taking place during the fire, then indicate how we "model" that process or set of processes.

Briefly: we have a fire ignited at a point in the center of a (horizontal) fuel slab. The heat from the flame pyrolyzes more of the material, and these hot, flammable vapors rise into the flame zone because of their buoyancy, and there ignite, maintaining the fire. The rising gas entrains some of the surrounding air, and the mixture forms a plume which rises to the ceiling. There it spreads out and forms a hot, sooty layer. If there is enough fuel, this layer will thicken until it comes below the top of the frame (called the soffit) of the highest vent, whereupon buoyancy will then drive some of the layer gases out of the room. The layer also loses energy by radiation and convection to the ceiling and walls. Provided there is enough fuel and it burns fast enough, flammable objects in the room will eventually reach their ignition temperature, and may ignite. When these ignitions begin to occur, they follow one another very rapidly, until all flammable objects are indeed burning. The transition period between ignition of the first "target item" and full room involvement we refer to as "flameover;" it generally only lasts a few seconds. Often flameover accompanied by, or closely followed by, "flashover", defined here as ignition of the layer gases, with flames emerging from the room of origin through one or more of its vents.

In the following paragraphs, I shall discuss these processes in more detail.* Implicit in the description above is the recognition that, although in principle the fire is one very complex phenomenon, we can analyze it by "dissecting" it--i.e., by successively focussing our attention on various features of the phenomenon (indeed, as soon as we distinguish one "feature" from the rest of the context, we have performed part of this dissection!). Thus, we assume that we can adequately--even accurately--describe the fire by a set of individual subprocesses with interactions among them.

A. The Fire.

We consider three possible kinds of fire: a burner fire, a pool fire, and one growing on a slab.

The burner is the simplest, as both its radius and the (gaseous) fuel flow rate are externally prescribed, are generally constant, and are independent of conditions in the enclosure.

The pool fire has, by definition, a constant area. We generally restrict ourselves to circular pools. However, the heat per unit area received by the fuel serves to gasify (evaporate or pyrolyze) fuel. The heat transfers to the fuel surface are by convection and radiation from the flames, from the walls and ceiling of the room, and from the hot gas layer in the room. Since these

*For still greater detail, see T.R. 34 (reference (3)). Also chapters II and III in reference (15).

vary with time, the pyrolysis rate from the pool also varies.

Finally, we have the growing fire: here, ignition is over a small area, but the heat transfers to the as yet uninvolved solid serve to heat it sufficiently that the fire grows. Although many theories of fire spread exist, none is adequate to predict the fire growth from first principles. Fire spread tests show the rate to be controlled by the fuel's surface temperature. However, CFC V uses a semi-empirical formula based upon open-air burning tests of the given fuel item.

These tests have consistently shown(17) that the pyrolysis rate and radius grow exponentially with time, between 80 and 170 sec. For earlier times, it grows more rapidly (as a power law; see pp. 77-8 of reference 18), but we ignore that and therefore use an effective initial radius which is larger than the real radius, but consistent with exponential increase from the beginning.

Our formula specifies a relationship between spread rate $\dot{R}$ and the net impinging heat flux. The heat flux to the surface normally consists of convection and radiation from the flame; for large luminous flames the latter is the more important, by far. In an enclosure, the heat flux is augmented by radiation from the hot layer and the hot ceiling (and walls); this feedback accelerates the burning rate, sometimes by a large factor, and our expression gives this. Again, although an expression for the convective heat flux from the turbulent flame back to the pyrolyzing surface can be written down [see, for example, de Ris(19)], we prefer to use a simpler expression in this version: for large flames, convection is negligible, so that the net flux is the difference between the external radiation and the reradiation from the burning surface. For a small flame, in contrast, radiation from the hot surface is considerably greater than the radiative flux to the surface from the flame, so that the surface would cool and the fire be quenched, but for the convective heat flux from the flame. We therefore add a convective contribution which falls with the radius at a rate such that the observed growth rate of the fire is maintained. This is *ad hoc*, but no more so than the assumption made in previous versions (CFC III and IV) that convection *just* cancels reradiation at all times.

The flame itself is generally luminous, turbulent, inhomogeneous, and anisotropic. We have nevertheless had surprising success so far, in modeling it as a grey, homogeneous cone. Indeed, for some of the radiation formulae we have gone even further, and assumed it to be a *point* source at the radiation centroid! For Mark V, we permit the vertex angle (2ψ) of this cone to increase appropriately, when the flame height is reduced by oxygen starvation or because the fuel nears burnout. The temperature and absorptivity of the flame gases is obtained by experiment. The assumed structure of the flame then determines the fraction of the heat released which emerges as radiation; the maximum value is empirically determined: for the flexible polyurethane foam which we have principally used in our experiments (P.U. #7004), this value is about 0.43.

Finally, in all the above cases, the efficiency of combustion--that is, the fraction χ of the heat of combustion H_c which is actually released in the fire--is empirically determined, too. For P.U. #7004, it is about 65%.

B. The Plume.

The only plume model we have used is that of Morton, Taylor, and Turner(20); detailed descriptions are given in T.R. 34 and in part V of this report. As in most of the rest of the physical processes involved, we assume that the mass and energy transfers from the fire to the layer is instantaneous--

i.e., there are no plume transients. The Morton, Taylor and Turner model also assumes a point source of heat, and the Boussinesq approximation (weak buoyancy). We assume a virtual point source below the fuel surface, at a distance such that the plume radius at the surface is just the fire radius. For the growing fire, of course, this distance also grows appropriately.

C. The Hot Layer.

Up to now we have used a two-layer model: the lower part of the room has ambient air only, while the upper part contains the hot, sooty layer, also assumed to be uniform in all of its properties--temperature, species concentration, absorptivity, etc. Again, this is not completely realistic, but it permits enormous theoretical simplification, and has worked reasonably well.

The layer receives mass and energy from the plume(s), and loses same through the vent(s). It also gains energy by absorption of some of the radiation from the flame(s), exchanges energy with the walls and ceiling by convection and radiation,* and loses energy by radiation to the floor, out the vents, etc.

The concentration of O_2, CO, CO_2, H_2O and "smoke" are calculated based upon empirical small-scale tests [see Tewarson(21)]. The absorptivity of the layer is provided in one of two forms. The preferred (default) form gives κ as a simple function of the smoke concentration. It is assumed to be grey, and the effects of CO_2 and H_2O molecular band absorption are ignored. A more elaborate program which may be chosen by the user at run time calculates κ from the concentration of CO_2, H_2O, and smoke [see Modak(22)].

*Energy is lost during the growth phase, gained when the fires are dying out or gone.

D. The Vents.

The flows through each vent in the room are calculated separately, taking the appropriate pressure at the center of the floor and using a hydraulic treatment for the thermal expansion and buoyant flow through the vent for that pressure (with an empirical flow coefficient). The appropriate pressure is determined by demanding that the net flow satisfy mass conservation.

E. Heat Transfer by Radiation.

There is radiative heat transfer between all bodies which can see each other, including flames, and radiative absorption and emission by intervening material. All these exchanges--among walls, ceiling, hot layer, flames, fires, and targets--are computed at present in different subroutines, in various (roughly equally accurate) approximations.

F. Heat transfer by convection.

Convective heat transfer between the hot layer and the extended ceiling,* between the outside (of the room) surface of the heated walls and the outside ambient air, and between the hot or cold layer and "target" objects, is calculated.

G. The heating of walls and "target" objects.

The net heat received at the surface, from radiation and convection, is taken to enter a solid according to the one-dimensional non-steady heat conduction equation, which is solved on an automatically set space grid through the thickness of the material: this yields the temperature profiles through objects. This is done in subroutine TMPO∅2. We also include TMPO∅1, an approximate calculation which was originally derived from the time-dependent solution for an infinitely thick solid exposed

*The "extended ceiling" means the ceiling plus the upper part of the side walls covered by the hot layer.

to an exponentially increasing heat source. (TMPOØ2 is the default version.) The same basic equations are used for the walls; however, in order to avoid having a wall temperature profile which varies as a function of height, it has been assumed that as the hot layer deepens, the newly exposed section of wall instantaneously absorbs enough energy from the layer to give it the same internal profile as the rest of the hot wall. Of course this is incorrect, as too much energy is thereby removed from the layer. We compensate for this error by taking <u>half</u> that energy, instead. The resulting differences are in fact very small. This calculation is made in subroutine TMPWØ1.

This completes the description of the physics involved. Next, we turn to the solution of the resulting equations.

III. NUMERICS.

The equations which describe each of the phenomena discussed above have to be written down and solved simultaneously. For Mark V, there are 15 variables associated with each room:

ROOM:	E_L	= energy of the layer
	$\dot{E}_L$	= time rate of change of layer energy
	m_L	= mass of the layer
	$\dot{m}_L$	= time rate of change of layer mass
	$\dot{E}_{LR}$	= net power gain of layer <u>via</u> radiation
	$\dot{E}_{LD}$	= net power gain of hot layer <u>via</u> convection
	h_L	= depth of the layer
	T_L	= temperature of the layer
	$y(O_2)$	= mass concentration of oxygen in the layer
	$y(CO)$	= mass concentration of carbon monoxide in the layer
	$y(CO_2)$	= mass concentration of carbon dioxide in the layer

	$y(H_2O)$	= mass concentration of water in the layer
	$y(S)$	= mass concentration of smoke in the layer
	Δp	= pressure at the center of the floor (relative to ambient)
	κ	= absorption coefficient of the hot layer

Each object in the room involves 14 variables:

OBJECT:	m_f	= mass of the object
	$\dot{m}_f$	= rate of increase of object mass (negative)
	$\dot{E}_f$	= negative of power release by combustion of its fuel vapors
	T_S	= surface temperature of object
	ϕ_{LO}	= radiative flux from the layer
	ϕ_{WO}	= radiative flux from the walls/ceiling
	ϕ_{FO}	= radiative flux from all flames
	h_p	= height of the lower part of the plume above the object*
	$\dot{m}_p$	= rate of mass transport from the plume into the layer
	$\dot{E}_p$	= rate of energy transport from the plume into the layer
	$\dot{E}_{pr}$	= power loss from the flame by radiation
	R	= radius of the base of the flame
tan ψ		2ψ is the vertex angle of the cone modeling the flame when the object is flaming.
	$\dot{m}_{au}$	= rate of entrainment of mass from the layer into the upper part of the plume. Taken to be zero in Mark V.

*We often refer to "the plume" as that part of the upward-flowing gases associated with the fire, lying between the fire base and the interface with the hot layer. In fact, of course, the plume continues up to the ceiling, although its character must change. Although in this version we do not yet include this upper part of the plume (between the interface and the ceiling), we have set aside a variable for its future inclusion (see $\dot{m}_{au}$).

Each vent in a room wall involves 3 variables:

VENT: $\dot{m}_u$ = mass (out)flow rate of hot (layer) gases

$\dot{m}_d$ = mass (out)flow rate of ambient air

$\dot{E}_u$ = energy outflow rate convected by hot gases

Each "wall" of a room involves 2 x 4 = 8 variables:

WALL: T_W = surface temperature of the wall

ϕ_{LW} = radiative flux from hot layer to wall

ϕ_{FW} = radiative flux from flames to wall

ϕ_{LWD} = convective heat flux from hot layer to wall

Each of these has <u>two</u> values, since the wall has two sides.

Thus the total number of variables--and hence equations-- is

$$N = 15r + 14o + 3v + 8w \tag{1}$$

where of course r is the number of rooms, o that of objects, v the number of vents, and w the number of walls. The "standard room" that we deal with has one "wall" (the extended ceiling), two objects (the initially ignited slab, and a target), and one vent (a doorway). Hence $r = 1$, $o = 2$, $w = 1$, $v = 1$, and $N(S.R.) = 15 + 2 \times 14 + 3 \times 1 + 8 \times 1 = 54$.

Thus, our "model" consists of a set of N coupled simultaneous equations in N time-dependent variables. Most of the equations are linear or non-linear algebraic equations. There is only one partial differential equation (for the diffusion of heat into a solid). All the rest of the D.E.'s trivial linear ordinary differential equations in t.

Consider the latter, first: We obtain, for example, the variable $\dot{m}_L(t)$ (the time rate of change of the hot layer mass) from a mass conservation equation. Since we also need $m_L(t)$, we "solve" the differential equation

$$\frac{d}{dt} m_L(t) = \dot{m}_L(t) \tag{2}$$

The solution is of course utterly trivial:

$$m_L(t) = \int_0^t \dot{m}_L(t')dt' \tag{3}$$

The <u>numerical</u> solution is found by using

$$m_L(t) \cong m_L(t - \Delta t) + \frac{\Delta t}{2}\left[\dot{m}_L(t - \Delta t) + \dot{m}_L(t)\right] \tag{4}$$

This is the trapezoidal rule; the inherent error is $-\frac{(\Delta t)^3}{12}\dddot{m}_L(\tau)$ in this time-step*(where $t - \Delta t < \tau < t$), so that the total error (assuming uniform time-steps only) is $\frac{(\Delta t)^3}{12} \sum_{t=i}^{n} \dddot{m}_L(\tau_i)$, where $n = t/\Delta t$. If we approximate the sum by an integral, we evidently have a cumulative numerical error

$$\text{Error} \cong -\frac{(\Delta t)^3}{12}\left(\ddot{m}_L(t) - \ddot{m}_L(0)\right). \tag{5}$$

* Provided that m(t) and its first three time derivatives are continuous. If $\dot{m}(t)$ is discontinuous, the error will be proportional to Δt.

2. Methods of Solution

At present, we use two methods of solution for this set of equations, both of them iterative. First, a successive substitution method; if and when that fails, we use the more powerful multivariate Newton-Raphson technique. The following is a detailed discussion of these techniques.

The principal method of solution is the successive-substitution method. This means that the variables x are calculated iteratively: after k iterations, the (k+1)st value of variable x is found by writing

$$x_i^{(k+1)} = g_i(\vec{x}), \tag{6}$$

where $\vec{x} = \{x_i\}$ is the array (loosely referred to as a "vector") of values previously found. Each of the variables is a function of t, but that explicit t-dependence is not shown, in order to make the equations more succinct. The solution represented symbolically by $\vec{x} = \vec{g}(\vec{x})$ is understood to be the solution at some given moment t. On the other hand, there is generally <u>no</u> explicit t-dependence in any of the functions g_i. If we take all the x_i on the right hand side of eq. (5) to be $x_i^{(k)}$--that is, the values found in the previous (kth) iteration--then we have the <u>Jacobi</u> variant of the successive substitution method. It is the one which has been used in all versions of the program up through CFC IV, and is handled in subroutine JACB. In CFC V, we use the Gauss-Seidel method, instead (the subroutine name has not been changed, however); in this variant, we always use the <u>most recently found</u> value of a variable. Thus, if we solve for the variables x_i in succession, we have* eqs.(7):

*Starting with eq. (7) and thereafter, we drop the parentheses about the index showing the iteration number, for simplicity. It should be clear from the context that the kth <u>power</u> of x_i is not meant.

$$
\begin{aligned}
x_1^{k+1} &= g_1\left(x_1^{k}, x_2^{k}, x_3^{k}, \ldots, x_n^{k}\right)\\
x_2^{k+1} &= g_2\left(x_1^{k+1}, x_2^{k}, x_3^{k}, \ldots, x_n^{k}\right) \qquad (7)\\
x_3^{k+1} &= g_3\left(x_1^{k+1}, x_2^{k+1}, x_3^{k}, \ldots, x_n^{k}\right)
\end{aligned}
$$

We can represent the successive substitution method for one variable,

$$x^{k+1} = g(x^{k}), \qquad (6a)$$

geometrically. Refer to figs. 1a-d. We always start with x_o as our initial guess.

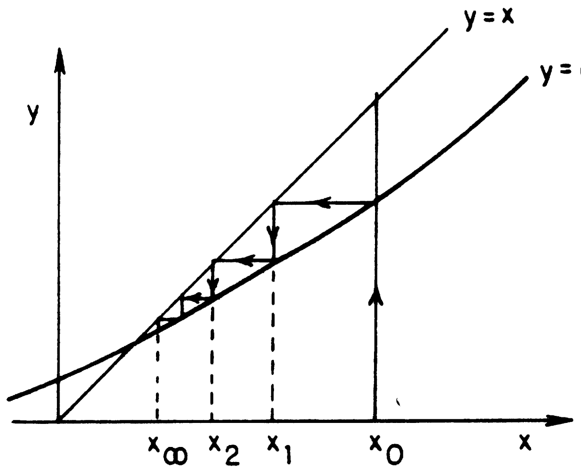


Fig. 1a. Geometric illustration of successive substitution method. x_0 is the starting trial value. Note that $g'(x)<1$.

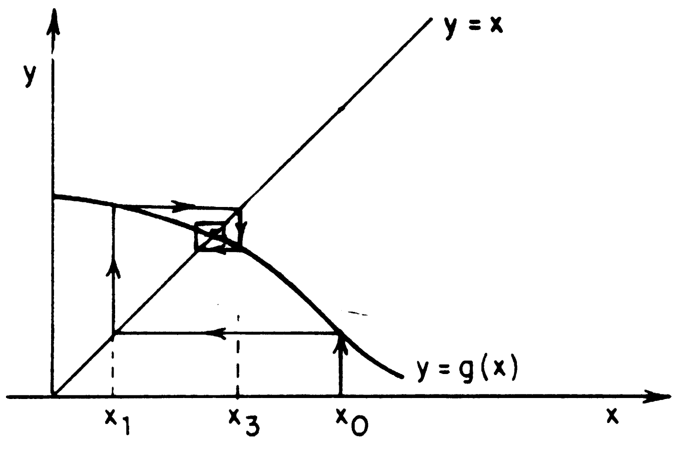


Fig.1b. In this case, $g'(x)<0$, but $|g'(x)|>1$.

In figures 1a and 1b we get convergence; for the latter case ($g'<0$) we evidently have the successive iterands oscillate about the correct answer.

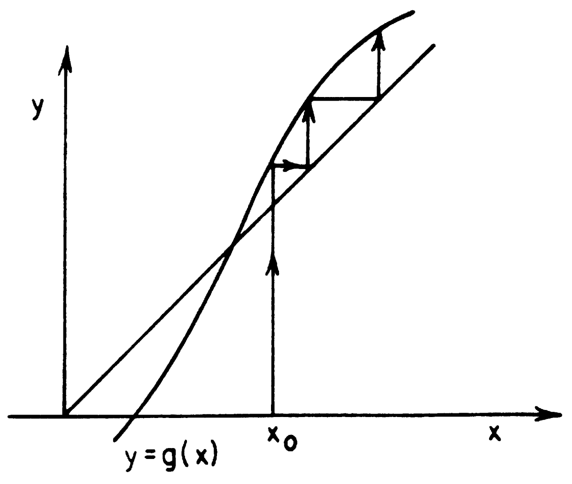


Fig. 1c. Same as fig. 1a, but $g'(x)>1$.

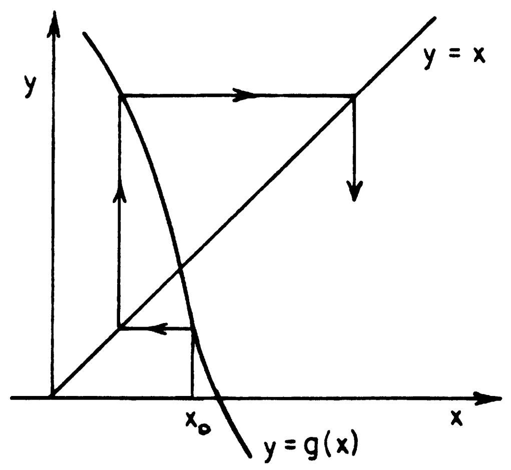


Fig. 1d. Same as fig. 1b, but $|g'(x)|>1$.

In figures 1c and 1d we see two cases where we would <u>not</u> get convergence; in the latter case ($g'(x)<-1$) we get diverging oscillations. It is thus clear from the geometry that we can only get convergence when

$$|g'(x)|<1 \tag{8}$$

in the region (x,x_0).

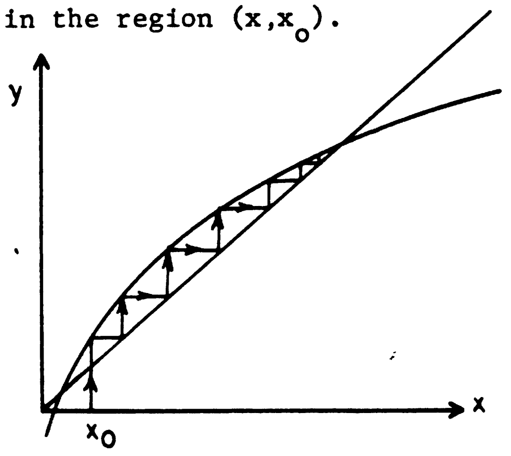


Fig. 1e. Convergence to a wrong root.

A more insidious problem occurs when we have $g'(x)>1$ in the vicinity of the desired root, but $g'(x)<1$ in the vicinity of an incorrect one, as shown in fig. 1e. For then, if the starting value is on the "near" side of the root, we will get convergence, but to the wrong value!

An important feature of Jacobi is that the results are quite independent of the order in which the equations are solved. This is clearly not the case for Gauss-Seidel, and the result is that some orderings will require more iterations than others, for convergence. Indeed, for some cases one ordering may diverge, while another ordering converges.

The Gauss-Seidel method* has worked much better for us than the Jacobi mehtod: it converges over a slightly wider domain, and generally far fewer iterations are needed. This method has been further refined and made more robust by using a damping factor: let us define

$$\Delta^k x_i \equiv g_i(\vec{x}^k) - x_i^k, \qquad (9)$$

where $$\vec{x}^k \equiv x_1^{k+1}, x_2^{k+1}, x_3^{k+1}, \ldots x_{i-1}^{k+1}, x_i^k, x_{i+1}^k, \ldots . x_n^k. \qquad (10)$$

Then Gauss-Seidel yields

$$x_i^{k+1} = x_i^k + \Delta^k x_i. \qquad (11)$$

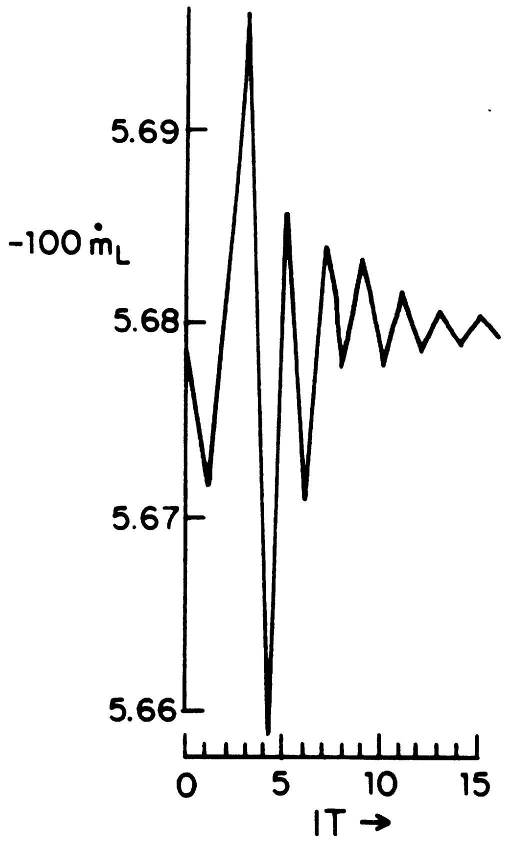
However, it is found that for many variables, successive iteration values oscillate about the final value, even as they converge to it. See, for example, figure 2a. This "overshooting" phenomenon can be reduced by taking only a fraction of the "correction" $\Delta^k x_i$:

$$\left(x_i^{k+1}\right)' = x_i^k + \lambda_i \Delta^k x_i.$$

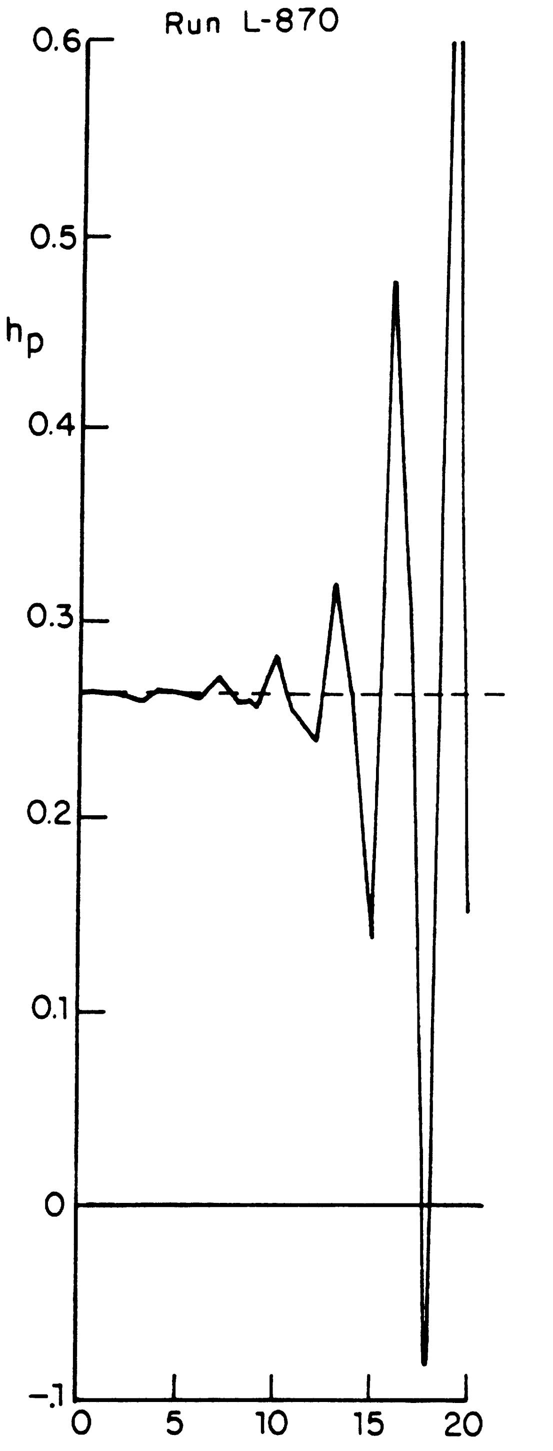
λ_i^{-1} is called the "damping factor" for that variable. Fig. 2a shows the slow oscillatory manner in which some variables converge by Gauss-Seidel without a damping factor. This procedure

* With one minor modification; see section 3, "Inputs and Outputs in the Physical Subroutines."

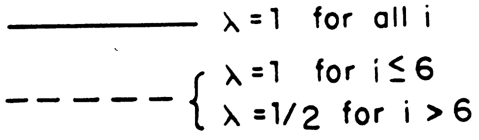
Fig. 2. Selected examples of how variables converge (or diverge!) as a function of the iteration number, IT.

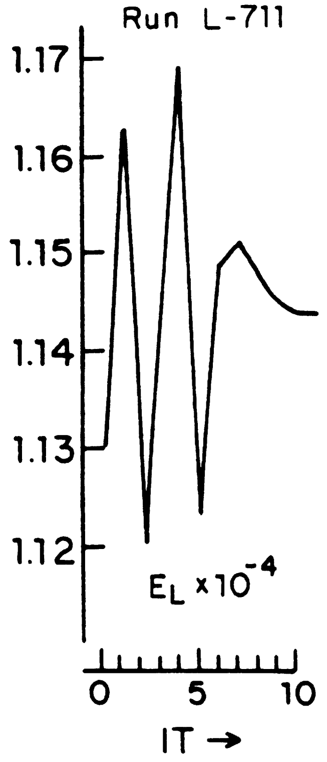


(a) From run L-733 (2/15/80)
At t = 260 sec.



(b) From run L-870, at t=300





(c) At t=260. $\lambda = 1$ for $i \leq 6$, $\lambda = 1/2$ for $i > 6$.

will sometimes lead to the convergence of an otherwise divergent series. Thus in figure 2b, the diverging osci ıtions of the variable h_p (plume height) are clearly in evidence; by switching to $\lambda = 1/2$ after the sixth iteration, this variable converged quite dramatically, as shown by the dashed line: the variable is completely converged by IT = 12. Figure 2b is somewhat misleading, actually: although still more unstable variables are tamed by this procedure, its power is nevertheless limited, of course. A more typical example is shown in Fig. 2c. The variable illustrated there, incidentally, TELZZ = $\dot{E}_L$, has generally been the most sensitive one, because (as seen in subroutine LAYR) it is the generally small difference between large numbers. We will come back to this in the section on convergence criteria.

Some numerical "experiments" showed that the best overall results are obtained by taking the same damping factor for all the variables, i.e., $\lambda_i = \lambda$. We also do best if we carry out the first <u>three</u> iterations with $\lambda = 1$, followed by $\lambda = 1/2$ for all subsequent iterations. Note that taking $\lambda = 1/2$ means that [from eqs. (11) and (9)]

$$\left[x_i^{k+1}\right]' = x_i^k + 1/2 \left[g_i\left[x^k\right] - x_i^k\right] = \frac{x_i^k + g_i\left[x^k\right]}{2} = \frac{x_i^k + x_i^{k+1}}{2} . \qquad (12)$$

That is, we take the arithmetic mean of the current and the previous iterations. In versions III and IV of the CFC, we iterated 100 times; if convergence did not then ensue, we switched immediately to a more powerful algorithm--the Newton method, to be discussed shortly. In the present version, if convergence does not occur in 35 iterations, then instead of immediately going to the Newton method, we halve the time interval, instead (to $\Delta t = 1$), and start over. If it then succeeds in converging, we run along with $\Delta t = 1$ for a number of steps (that number, to be described shortly), then double the time increment, since the difficulty may have only been temporary. On the other hand, if it still does not converge in 35 steps with $\Delta t = 1$, we halve again. And so on. If this procedure does not produce convergence by the time that $\Delta t = 1/8$, <u>we then</u> switch to the Newton method, with $\Delta t = 1/8$. In this version, moreover, we try to switch back to Gauss-Seidel after a reasonable time, as will be described later in this section.

In any iterative procedure, we must have starting values (for each time t). And in order for the procedure to converge, these must lie within the "circle" of convergence (the N-dimensional volume in variable space, actually) corresponding to this method. Moreover, unless we are in an unstable region (where the circle of convergence has shrunk to a <u>point</u>), the closer we are to the correct solution, and the faster we will converge.

We assume that all the physical variables are continuous functions of time. We then have available at the outset of the solution at time $t+\Delta t$ a very good initial guess for $\vec{x}(t+\Delta t)$: namely, the previous solution vector $\vec{x}(t)$--or better, new values extrapolated in time from $\vec{x}(t)$ and $\vec{x}(t-\Delta t)$. Moreover, the accuracy of such an intitial guess may be made arbitrarily good by cutting the time step sufficiently. For version V, we use the (converged) solutions for the <u>three</u> previous time steps to carry out what is essentially a quadratic extrapolation for each variable, in order to have good starting values. That is referred to as the "zeroth" iteration (IT = 0). Our basic time increment has been $\Delta t = 2$ sec, which is large enough to permit solution of the overall fire problem in a reasonably short time, yet small enough to introduce only relatively small numerical errors.

When the method converges, it converges linearly. That is, for sufficiently large k (i.e., $k > k_c$).

$$x_i^{(k+1)}-x_i^{(k)} = C_i\left[x_i^{(k)}-x_i^{(k-1)}\right] \tag{13}$$

for all $k>k_c$, with $|C_i|<1$. This holds for Gauss-Seidel as well as for Jacobi. If $|C_i|\geq 1$ for any i, however, then it does not converge overall. The overall convergence rate evidently goes as $\max\{|C_i|\} \equiv C$. We can then also write, symbolically,

$$|| x^{k+1}-\vec{x} || = C|| \vec{x}^k-\vec{x} || \tag{14}$$

where $\vec{x}$ is the exact solution (at time t). Then for C<1, the algorithm will yield convergence to the solution, taking roughly a constant number of iterations for each successive increase in accuracy by one decimal place. If only constants C>1 can be found for all values of Δt, the algorithm will fail, and the vector will tend to diverge at the same steady rate. It is easy to show analytically that C is the maximum value of $|g'(x)|$ in the interval covered by the iterations, as was indicated by figs. 1a-d, and given by eq. (8).

The second method used for the solution of the equations is the multivariate Newton-Raphson technique [see e.g. reference (23), §71-3, 80]. Whereas the Jacobi method is directed to solving a system of the form $\vec{g}(\vec{x}) = \vec{x}$, a Newton method works on a system of the form $\vec{f}(\vec{x}) = \vec{0}$.

Therefore, we begin implementing any Newton kind of method by defining the vector function $\vec{f}$ as follows:

$$\vec{f}(\vec{x}) \equiv \vec{x}-\vec{g}(\vec{x}) \qquad (1$$

In one variable, the Newton-Raphson method assumes that f(x) is linear; so that if at the kth iteration the value $x^{(k)}$ is not a solution--i.e. $f_k \equiv f(\vec{x}^k) \neq 0$--then the "correct" value is given by

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)} \qquad (1$$

The method can be visualized as a process of extrapolating from each approximation to the next along a tangent line:

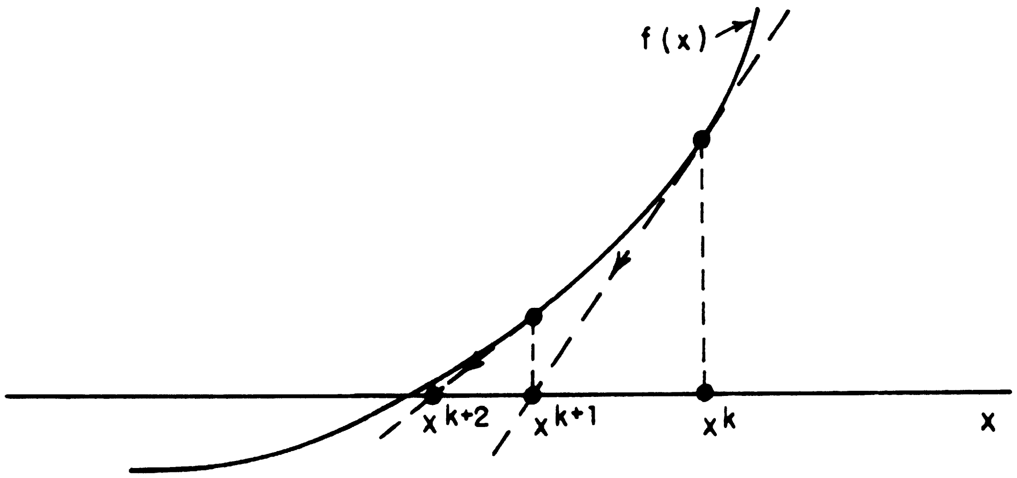


Fig. 3. Illustration of the geometry of the Newton-Raphson Method.

In this example, the process will (clearly) converge rapidly. Note: Using eq. (15) in one dimension, we can write eq. (6a), which describes the succesive-substitution method, as

$$x^{k+1} = x^k - f(x^k).$$

(Note that for a single variable, there is no difference between Jacobi and Gauss-Seidel). Comparing that with eq. (16), it is clear that the two methods would be the same if $f'(x) = 1$.

There are two general cases where the Newton technique will fail to converge: first, if we are too far from the desired root for the initial guess, we might--at best--converge to an incorrect root, as is clear from fig. 4:

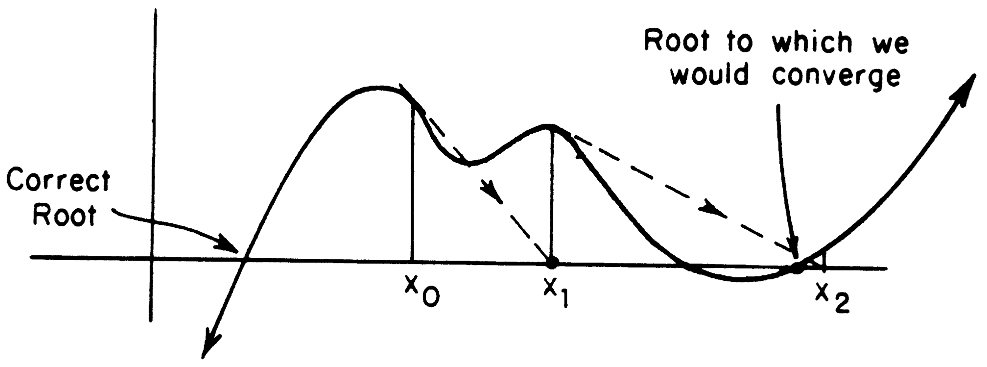


Figure 4. Example of convergence to an incorrect root, when we start the Newton process outside the region of convergence.

That is, we start outside the region of convergence. Second, as is clear from eq. (16), it will fail if $f'(x^k) = 0$ for some k. Indeed, even if the derivative is not precisely zero, but merely very small, it will take the next iteration outside the region of convergence.

It is not difficult to show [see ref. (23), § 78] that a sufficient condition for convergence is

$$|f'(x^k)|^2 > |f(x^k) f''(x^k)| \tag{17}$$

for all k.

The Newton-Raphson technique can be generalized to the case of n simultaneous equations (in n variables). Then eq. (16) becomes

$$\vec{x}^{k+1} = \vec{x}^{k} - \mathcal{J}^{-1}\vec{f}(\vec{x}^{k}) \tag{18}$$

where $\mathcal{J}$ is the n-by-n Jacobian matrix $\left\{\frac{\partial f_i}{\partial x_j}\right\}$:

$$\mathcal{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & - - - & \frac{\partial f_1}{\partial x_n} \\ - - - & - - & - - - \\ - - - & - - & - - - \\ \frac{\partial f_n}{\partial x_1} & - - - & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{19}$$

If we write out the inverse of $\mathcal{J}$ in terms of its cofactors, we find that for the ith variable, eq. (18) yields

$$x_i^{k+1} = x_i^{k} - D^{-1} \left| \begin{matrix} \frac{\partial f_1}{\partial x_1} & - - - - & \frac{\partial f_1}{\partial x_{i-1}} & f_1 & \frac{\partial f_1}{\partial x_{i+1}} & - - - - & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & - - - - & \frac{\partial f_n}{\partial x_{i-1}} & f_n & \frac{\partial f_n}{\partial x_{i+1}} & - - - - & \frac{\partial f_n}{\partial x_n} \end{matrix} \right|_{\vec{x} = \vec{x}^{(k)}} \tag{20}$$

where $$D \equiv \det(\mathcal{J}) \tag{21}$$

To implement eqs. (18) or (20) numerically, we need the Jacobian matrix $\mathcal{J}$ at each iteration. It is not practical to determine $\mathcal{J}$ directly by computing partial derivatives analytically, so we use a finite difference scheme. To compute column j of $\mathcal{J}$ we perturb $\vec{x}^{k}$ by the small amount h in the j th position. The perturbed vector may be written $\vec{x}^k + h\vec{e}_j$, where $\vec{e}_j$ represents the unit vector with 1 in position j and zeros elsewhere. We then set

$$J_{ij} = \frac{f_i\left(\vec{x}^k+h\vec{e}_j\right)-f_i\left(\vec{x}^k\right)}{h}, \quad i = 1, \ldots, n. \tag{22}$$

We have taken $h = \delta \cdot x_j^k$ with $\delta = 10^{-4}$ in previous versions of the program. This occasionally has led to trouble, and in this version we use $\delta = 10^{-3}$, which seems to work more satisfactorily. (See comments under the Scaling of Variables section). Computing the full matrix $\mathbb{J}$ means going through eq. (22) n times. Once $\mathbb{J}$ has been estimated, we do not directly compute its inverse, as eq. (18) would suggest. Instead, we rewrite eq. (18) in the form

$$\mathbb{J}\left(\vec{x}^k-\vec{x}^{k+1}\right) = \vec{f}\left(\vec{x}^k\right) \tag{23}$$

and solve this linear system for $\vec{x}^k-\vec{x}^{k+1}$, from which $\vec{x}^{k+1}$ can be determined, since $\vec{x}^k$ is known. The linear system is solved by Gaussian elimination with partial pivoting (ref. (23), p. 270 ff) using the code fo Forsythe and Moler (25). The algorithm of Gaussian elimination used in solving this system begins by decomposing $\mathbb{J}$ into the product of a lower-diagonal matrix $\mathbb{L}$ with 1's along the diagonal and an upper-diagonal matrix $\mathbb{U}$; performing this L-U decomposition takes about $n^3/3$ multiplications (ref. 26). The computational time required by the Newton method is used primarily in two steps of the algorithm: in n evaluations of $\vec{f}$ to compute $\mathbb{J}$, and in solving the linear system (23). These jobs make one iteration of the Newton method much more time-consuming than one iteration of the Jacobi (or Gauss-Seidel) method.

On the other hand, Newton methods often will converge with far fewer iterations than the successive substitution methods: if an exact Jacobian were known at each iteration, then the algorithm given by eq. (18) would converge not linearly but quadratically, i.e. for all k greater than some critical value k_c,

$$\| \vec{x}^{k+1} - \vec{x} \| \leq C \| \vec{x}^{k} - \vec{x} \|^{2} \qquad k > k_c \tag{2}$$

(c.f. equation (14) for the Jacobi method).
The quadratic dependence means that a reasonably good initial guess $\vec{x}^0$ is very important, but that fast and reliable convergence may be expected when one is available, as long as the n-dimensional generalization of the convergence criterion (17) is satisfied. Eq. (24) implies that convergence will take roughly a constant number of iterations for each doubling in the number of significant digits.

Our finite difference approximation to the Jacobian is not exact, so that in fact we are using a secant, rather than the tangent indicated in figure 3. It can be shown that this leads to a rate of convergence which is neither linear nor quadratic; instead, we should expect the norms to follow

$$\| \vec{x}^{k+1} - \vec{x} \| \leq C \| \vec{x}^{k} - \vec{x} \|^{1.839} \qquad k > k_c \qquad (25$$

for the secant approximation, in one dimension. See, for example, Jarrat, in reference (29). It is also shown there, incidentally, that the Newton method, in this approximation, is equivalent to quadratic interpolation.

Super Fast Newton Method (NWSF). The full Newton method is quite time-consuming, and alternatives were devised. The simplest technique is to recompute the Jacobian not at every iteration, but perhaps only once per time step. Indeed, if $\mathcal{J}(t)$ varies slowly enough, only once every few time steps. This works much of the time, and is indeed very much faster than the use of NWTN.

We also had a "fast Newton" subroutine, NWTF; this uses a compromise method which updates the Jacobian only partially at each iteration. Specifically, at the beginning of each iteration it examines the vector $\vec{f}(\vec{x}^k)$ to determine which component of f is largest in absolute value -- farthest from the desired value of 0. Suppose this component is f_j. NWTF then updates only the jth column of $\mathcal{J}$, using eq. (22). Thus only the partial derivatives with respect to variable x_j are recomputed, in the hope that this will reduce the magnitude of f_j substantially at the next iteration and so bring the system significantly nearer to a solution. In practice it was found, however, that NWTF was not much more useful than NWSF, although slower. It has therefore been dropped in CFC V, along with MODJ, its implementing subroutine.

The computation proceeds as follows: the calculation runs along in Gauss-Seidel until it fails to converge in 35 iterations. It then cuts the time step in half and starts over again in Gauss-Seidel, using newly extrapolated values from the previous time steps; this guarantees a starting solution closer to the correct answer, which generally leads to convergence. However, failure to converge in 35 of these smaller steps causes another cut in the time step. Until we manage to converge, the time step will repeatedly be halved; this will be repeated until

$$\Delta t \leq \Delta t_0/20 \qquad (26)$$

where Δt is the newly cut time step, and Δt_0 the originally chosen time step. At this point, we switch over to NWTN, with time step of Δt. With the canonical Δt_0 of 2 sec, we switch over to NWTN when we fail to converge with $\Delta t = 1/8$. We go ahead in NWTN, with this new $\Delta t = 1/8$th sec, for $t_0 = 2$.

Now, suppose we have had to switch to NWTN. If NWTN converges, we use NWSF for the next time step, with the same time increment. With one exception, it will then continue in NWSF with the same increment for a number of steps; when the _total_ number of time steps taken since the beginning of the program (NT) reaches a multiple of 10, the time increment is doubled (thus the exception is, when NWTN happens to converge at NT =10m - 2, one NWSF step is taken at NT = 10m - 1. and the _next_ one is NT = 10m--thus it takes just the one step with Δt, and then doubles Δt immediately).

The solution mode will remain NWSF until it ceases to converge (in which case it goes back to NWTN, and we start over), or until the size of the time step _just taken_ is $\geq \Delta t_0/4$, in which case it switches back to the Gauss-Seidel mode. Thus if NWTN converges for $\Delta t = 1/8$, the program will switch to NWSF with $\Delta t = 1/8$, continue that way through $\Delta t = 1/2$ (assuming success all the way), and switch back to Gauss-Seidel with $\Delta t = 1/2$. Again, it takes a number of (successful) steps in G.S. mode, until NT = 0 mod(10) (as usual), whereupon Δt is doubled. This doubling procedure continues until $\Delta t = \Delta t_0$ again, where it stays, or until trouble occurs again. On the other hand, if NWTN fails to converge with $\Delta t = 1/8$, it tries again with $\Delta t = 1/16$ (still in NWTN). If that _still_ fails to converge, the message "YOU ARE IN TROUBLE" flashes on the screen, and the built-in "debugger" is invoked, giving us the option of stopping the calculation, or of continuing to cut down the time increment. (Unfortunately, the latter procedure is generally useless.) When running in batch mode (see section IV, esp. IV, 7), an automatic program stop is produced, without invoking DEBUG.

There are two other cases when the time step is cut: we want printout at some integral multiples of a reasonable δt--generally, 10 seconds; if Δt is such that $t+\Delta t$ is greater than the next multiple, Δt is cut down to $\Delta t'$, the increment needed to make $t + \Delta t' = 10p$ (p integral). Again, when an object ignites, it of course does so within the current interval Δt -- say, it ignites at $t+\delta t < t+\Delta t$. In that case the next increment is $\delta t'$, where $\delta t+\delta t'=\Delta t$. In these two cases, where Δt is cut for printout or ignition, the original time step Δt is immediately restored.

Finally, we might point out that while in the Gauss-Seidel mode, we never take fewer than five iterations (even if the convergence criterion is satisfied for IT < 5), because the convergence criterion we use is far from infallible and we are prone to getting a spurious "convergence" for IT < 4. (We occasionally get a spurious convergence even with a large number of iterations!) There are a number of other minor restrictions and exceptions which we need not go into here.

3. Inputs and outputs in the physical subroutines.

The simplest way to proceed would be to have each use of an equation in the physical subroutines result in the new value replacing an old one for the corresponding physical quantity. For example, the equation yielding the heat flux from the layer to the ceiling is

$$\dot{q}''_{LW} = h_i(T_L - T_C) \qquad \text{(a)}$$

where h_i is the heat transfer coefficient of the layer gases, T_L the layer temperature, and T_C that of the ceiling (see subroutine CNVW, in section V). This equation could be realized as

```
FQLWD = HI*(ZKLZZ - ZKWZZ)                (b)
```

This value would then immediately be placed in VAR, and if an equation used subsequently contained FQLWD on the right-hand side, the value taken there would necessarily be the one just calculated by eq. (b). This is just the Gauss-Seidel prescription, in the successive substitution mode.

To allow more flexibility in varying the method of numerical solution, however, it is necessary to be able to manipulate equations like (b) more freely by separating inputs (right hand side of the equation) from outputs (left hand side). Except for the Gauss-Seidel variant, we do not want the output of each equation to become the new value of the corresponding physical quantity. For example, the finite-difference calculation of eq. (22) requires a determination of $g_i(\vec{x}^k + h\vec{e}_j)$, but by no means do we want this artificial quantity to replace the previous value of x_i. Only after the Jacobian matrix has been determined will a new x_i be determined by solving the linear system (23).

We separate the inputs and outputs as follows: the physical subroutines get their inputs x_i from (the appropriate places in) common block VAR. Each output variable has the suffix "1" added to its name to distinguish it from the corresponding five-character name appearing in /VAR/. Thus eq. (b) is written as

$$\text{FQLWD1} = \text{HI}*(\text{ZKLZZ} - \text{ZKWZZ}) \qquad \text{(c)}$$

This output could be put into the argument of subroutine CNVW; we have chosen, instead, to place these outputs into common block VAR or NEWVAR after each subroutine call, depending on whether we are using Gauss-Seidel or Newton method. This is discussed in greater detail in Appendix 6.2 of section IV.

It must be pointed out here that this procedure will sometimes prevent the Gauss-Seidel method from being fully implemented: when variable x_i is calculated in one part of a subroutine, and then used in a later part of the same subroutine to evaluate another variable x_i, it will be the <u>Jacobi</u> technique which is implemented within the subroutine. To the extent that this occurs, we actually have a hybrid algorithm, rather than "pure" Gauss-Seidel. But this happens for only a few variables, and the departure from G.S. is quite small.

4. Scaling of variables.

The physical variables in the computer fire code range in magnitude from energies on the order of 10^7 (Joules) to masses on the order of 10^{-5} kilograms, and even smaller quantities. For a variety of reasons, it is difficult to work numerically with sets of variables that range as these do through twelve or more orders of magnitude. One problem comes about as a result of the finite accuracy of the computer's floating point representation of numbers; for example, in estimating a partial derivative according to a finite difference $f(x + h) - f(x)$, the perturbation h cannot be too much smaller than x in magnitude when x is already small, or the machine will compute the resulting difference as exactly 0 or at least compute it inaccurately. Again, in order to have maximum accuracy in solving the simultaneous (linear) equations in the Newton mode, we need to have all the variables of comparable size. In previous versions, we did this (for all numerical modes) by multiplying each variable by some multiple (or submultiple) of 2 -- i.e. 2^{ni} -- in an "interfacing" subroutine SCAL. The logical flow of the program was complicated by this, however, and so we tried to drop scaling altogether. That gave poor results, but we have found good results by scaling only in the NWTN mode. We no longer use the multipliers 2^n -- instead, we use the value of the variable <u>itself</u>, as a scaling factor. Thus, whenever Newton is invoked, the variables are first all scaled to unity; then the derivatives for the Jacobian matrix are found. That is done as indicated by eq. (22).

5. Convergence

Our general convergence criterion is that two successive iterations do not change the value of any variable by more than some small value. Since we want the relative accuracy to be about the same for all variables, we normally would demand that

$$\left|\frac{\Delta x_i}{x_i}\right| < \varepsilon_c \qquad (27)$$

for all i, where $\Delta x_i \equiv x_i^{k+1} - x_i^{k}$. Now, when a variable takes on a very small value, such that it essentially ceases to have physical significance, we cease to test if for convergence. (The array of these minimum values is called VMIN, in the program, and appears at the end of the DATA BLOCK.) When a variable takes on a value greater than but comparable to its VMIN value, then of course we do not need the accuracy given by eq. (27). Indeed, an accuracy within a factor 2 is sufficient. Hence, eq. (27) is replaced by

$$\left|\frac{\Delta x_i}{x_i}\right| < \max\left[\frac{x_{i\ min}}{|x_i|}, \varepsilon_c\right] , \qquad (28)$$

where $x_{i\ min}$ is the VMIN value of variable i.

On the other hand, suppose variable j is the small difference of two large numbers. Then in order for x_j to be computed to within ε_c, its large constituents must be computed rather <u>more</u> accurately! This means that (a) some variables are computed much more accurately than ε_c, and (b) by the same token, if we take ε_c too small, truncation and round-off errors begin to be significant, and will make it literally <u>impossible</u> to get "convergence", so that ε_c must be judiciously chosen.

The value ε_c is called TOLER in the program, and is taken to be 3×10^{-4}. This value is small enough to give satisfactorily accurate answers, but large enough to permit convergence in a reasonable time.

Evidently if the largest value of $\left|\frac{\Delta x_i}{x_i}\right|$ is smaller than ε_c, then eq. (27) is satisfied for all i. Since we want to use the criterion (28) instead of (27), we define

$$\text{TEST}(i) \equiv \left|\frac{\Delta x_i}{x_i^k}\right| \cdot \frac{\varepsilon_c}{\max\left(\varepsilon_c, x_{i\,\min}/\left|x_i\right|\right)}$$

The largest value of TEST(i) is then the NORM at the kth iteration. (In Gauss-Seidel we refer to these as HNORMS; in Newton-Raphson, as FNORMS.) Thus we find the NORM at each iteration; when NORM < ε, we say we have converged.

In the G.S. mode, we (generally) take as many iterations as needed, up to 35. The NWTN mode is so much more time-consuming, however, that we want to take as few such iterations as possible. This means that we want to determine very quickly, whether we are diverging or not. This is done is subroutine CONV. We can only begin to make an intelligent guess as to whether the procedure is converging or not, when we have a minimum of three iterations. Since the zeroth iteration is the predictor, this gives us three FNORMS. We then examine them. If the norms are monotonically decreasing in value, we can make an estimate of how many more iterations are probably required to converge according to our criteria. This information is not used in the present version, however.

If the three norms all correspond to the same variable, and the most recent one is larger than the first two, it's a good bet that the procedure is diverging -- we halve the time interval, rather than (probably) waste time. If we've already halved the time increment once, however, it seems unlikely that we'll get convergence by halving the time interval again, with this method. (Still, the debugger is invoked at this point in order to give us various options, rather than just stopping the calculation, unless we are in batch mode.)

On the other hand, if the three norms do not all correspond to the same variable, we have insufficient information to be able to decide, again with the same Δt.

Finally, we note that there are islands of instability--although "trenches" or "black holes" may be more apt descriptions! Some of these will be discussed in section VI. Here we mention only the following: a 500-sec run converges smoothly and with no notable difficulty, taking (mostly) 2-sec. steps. But if we change the basic time step to $\Delta t=1$ sec, we run into trouble at $t=409$: we cannot converge at that value! Evidently this caused no difficulty with $\Delta t=2$, because we stepped over the "trench" which begins at 409, going from 408 directly to 410! This difficulty at $t=409$ remains until we get down to $\Delta t=0.125$ sec. Then a smaller trench appears, at $t=359.125$. This of course suggests a simple way to overcome difficulty at any point: try to step "over" it. This may do the trick, but sidesteps the issue (no pun intended).

6. Accuracy

In principle, the smaller we make ε_c, the convergence criterion, the more accurate the overall result should be. Similarly, the smaller we make Δt_o, the basic time increment, the smaller will be the errors arising from the use of numerical approximations of various sorts. There are two principal limitations, however. First, the practical one: the smaller we make ε_c, the more iterations have to be taken in order to converge; moreover, the more likely we are, in fact, to fail to converge at all. Indeed (as pointed out earlier), if we demanded too much precision we would run into truncation errors, and we could never converge. The second limitation is one of principle: for a given time step Δt_o, the (fractional, rms) numerical error is ε_n, which evidently is proportional to some power of Δt_o. For small ε_c and a large Δt_o, the numerical error dominates. For sufficiently small Δt_o, on the other hand, the principal error will be due to the finite ε_c. In the latter case, the total error at the end of n steps will then be $\sim n\varepsilon_c$ or $\sqrt{n}\varepsilon_c$, depending on whether the errors are systematic or random.

But for a given run length t, $n \sim t/\Delta t_o$, and hence the total error will <u>grow</u>. Thus there is an unavoidable minimum error for any run.

Let us now consider these errors in a little more detail. First, again consider ε_c. Our developmental work is now principally being done on a VAX 11/780. The VAX is a 32-bit machine--of these, about 24 bits (in single precision) are reserved for the mantissa, corresponding to 7 decimal places of accuracy. Thus setting $\varepsilon_c = 10^{-7}$ would result in immediate failure to converge. In fact, because at least one variable ($\dot{E}_L$) is the small difference of large numbers, 10^{-6} would be very troublesome. The value 3×10^{-4} was chosen after numerical experiments showed that the final answers (for a 500-sec run of the standard problem) did not differ by more than a few percent for <u>this</u> value of ε_c, compared to more accurate calculations (i.e., with smaller ε_c's). Larger values of ε_c resulted in deviations judged to be unacceptable in magnitude.

Next, consider the overall error per step. (I will omit the qualifier "fractional" in front of error, as understood from now on. Similarly, if a particular variable is not mentioned, "error" <u>means</u> "rms error"). For a convergence criterion ε_c, the mean convergence error will be $\beta\varepsilon_c$, with β of order unity, and the errors will be assumed to be random. If the numerical error ε_n is also random, and incoherent with the convergence error $\beta\varepsilon_c$, then the error per step will be

$$\varepsilon = \sqrt{\varepsilon_n^2 + \beta^2 \varepsilon_c^2} \tag{29}$$

and the total error after n steps,

$$\varepsilon \doteq \sqrt{n}\,\varepsilon \tag{30a}$$

On the other hand, if the numerical errors are systematic, then

$$\varepsilon \cong n\varepsilon_n + \sqrt{n}\beta\varepsilon_c \tag{30b}$$

If the basic time increment is Δt_o, then for a run of duration t,

$$n \geq t/\Delta t_o \tag{31}$$

(The inequality evidently applies, since we might be forced to halve an interval.) A series of runs with Δt_o = 2, 1, 1/2, 1/4, and 1/8 was made in order to investigate this point (with ε_c kept at 3×10^{-4}). It was found that all the variables appear to converge to asymtotic values. Indeed, the deviations from those values were very nearly linear with Δt_o:

$$x_i(\Delta t_o) \cong x_i(0) \left[1 + \alpha_i \Delta t_o\right] \qquad \text{all } i \tag{32}$$

Thus the error at time t is apparently

$$E(t) \cong \langle\alpha(t)\rangle \Delta t_o \ , \tag{33}$$

where $\langle\alpha\rangle$ is the rms value of the α_i's. (Only for Δt_o = 2 sec do the relationships (32) and (33) begin to fail.) Since the values still appear to be converging with Δt_o = 1/8, it appears that (a) the value of Δt_o for which we get minimum error is much smaller than 1/8th sec, and (b) $\beta\varepsilon_c$, which is independent of Δt_o, must be $\ll\sqrt{n}\varepsilon_n$ even for Δt_o = 1/8 (unless the convergence errors are systematic). Moreover, 80% of the α's in eq. (32) are negative and (mostly) have the same sign at t = 150 and at t = 300. A large sample of variables gave (for the standard run) the following rms. values:

$$\text{At } t = 150 \text{ sec,} \quad \langle\alpha\rangle \equiv \sqrt{\textstyle\sum_i \alpha_i^2/n} = 0.0145$$

$$\text{At } t = 300 \text{ sec}, \qquad \langle\alpha\rangle = 0.0168 \qquad (34)$$

For shorter times $\langle\alpha\rangle$ is still smaller, as expected. Thus a typical variable will behave as shown in fig. 5. (The errors are, of course, greatly exaggerated, in the interests of clarity.) Thus a third conclusion is that (c) the numerical error are in fact systematic rather than random--i.e. they are all in the same direction--and therefore eq. (30b) must be the correct expression.

Thus, we should be getting pretty reliable answers, with $\Delta t_o = 1/8$. On the other hand, the total computing time roughly doubles, for each halving of Δt_o. Since $\Delta t_o = 2$ seems to lead to <u>sufficient</u> accuracy (final errors of 5% or less), that has been chosen as the default value.

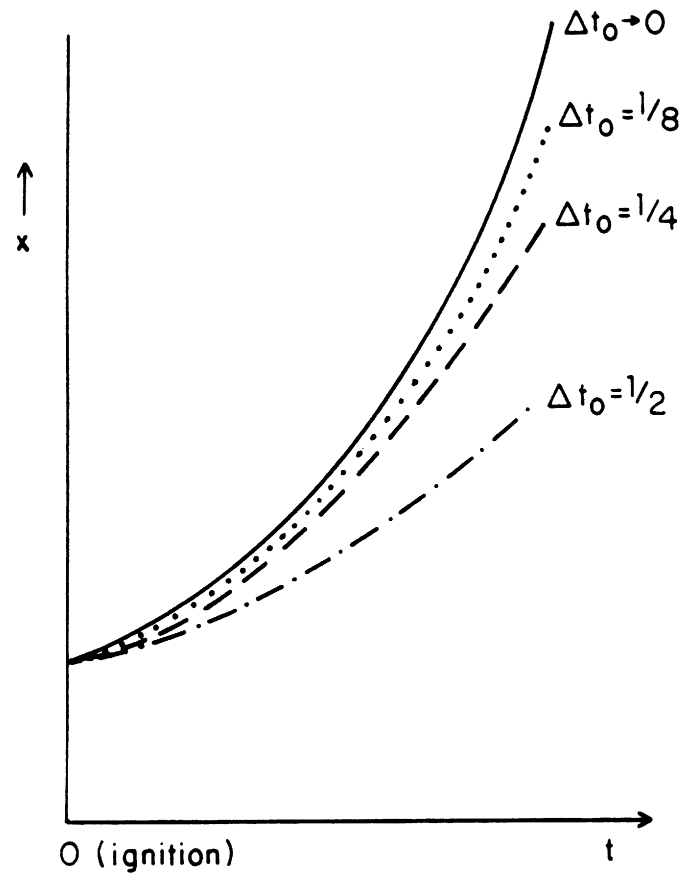


Fig. 5. The values $x(t)$ taken on by a typical variable, as found by numerical calculations made with time increments Δt_o of different magnitudes. The solid curve is the solution (which would be) found with $\Delta t_0 \to 0$. However, this solution is still not exact, since $\varepsilon_c > 0$.

One final point: from the above, we infer that

$$E(t) \cong \langle\alpha\rangle \Delta t \cong n\varepsilon_n + \sqrt{n}\beta\,\varepsilon_c \cong n\varepsilon_n \cong \frac{t}{\Delta t}\,\varepsilon_n\,,$$

[the last (approximate) equality comes from eq. (31)]. Thus

$$\varepsilon_n \doteq \frac{\langle\alpha(t)\rangle}{t} (\Delta t)^2 \tag{35}$$

Although we would expect from this equation that $\langle\alpha(t)\rangle \propto t$, whereas eq. (34) shows that that is not quite the case, eq. (35) does show that use of the trapezoidal rule for integration is not the principal source of numerical errors, since it varies as $(\Delta t)^3$.

7. Time required for a calculation

The time required evidently will depend on:

(a) the size of the problem (number of rooms, objects, vents, etc.),

(b) the complexity of the program,

(c) the speed and size of the machine,

(d) on a time-sharing system, how busy the machine is.

We use the "standard run" (SR) as a benchmark; this configuration is one room with a doorway, and two objects in it. Object #1 is burning, with a growing fire; #2 a target which ignites at $t \cong 310$ sec. The time required for a 500-sec. SR with Mark IV, on a PDP10, was about 7.5 to 8 min of CPU time.* This was reduced to something of the order of 80 to 120 sec on the VAX 11/780. In fig. 6 the CPU time τ required for runs of various lengths t is plotted. Curve A (marked "standard run") corresponds to the case with version 4, before the switch to Gauss-Seidel with damping, and where we switched directly over to NWTN when 100 iterations of successive substitution did not yield convergence. As we can see, the CPU time increased rapidly for $t>280$--roughly in synchrony with the exponentially increasing tempo of the fire. There is no sharp knee in the curve, as we might have expected upon target ignition at $t\sim310$.

Curve B shows the CPU time required for the same run, with the new

* CPU = Central Processor Unit--the "heart" of the computer, excluding Input/Output (I/O) time on the peripheral equipment.

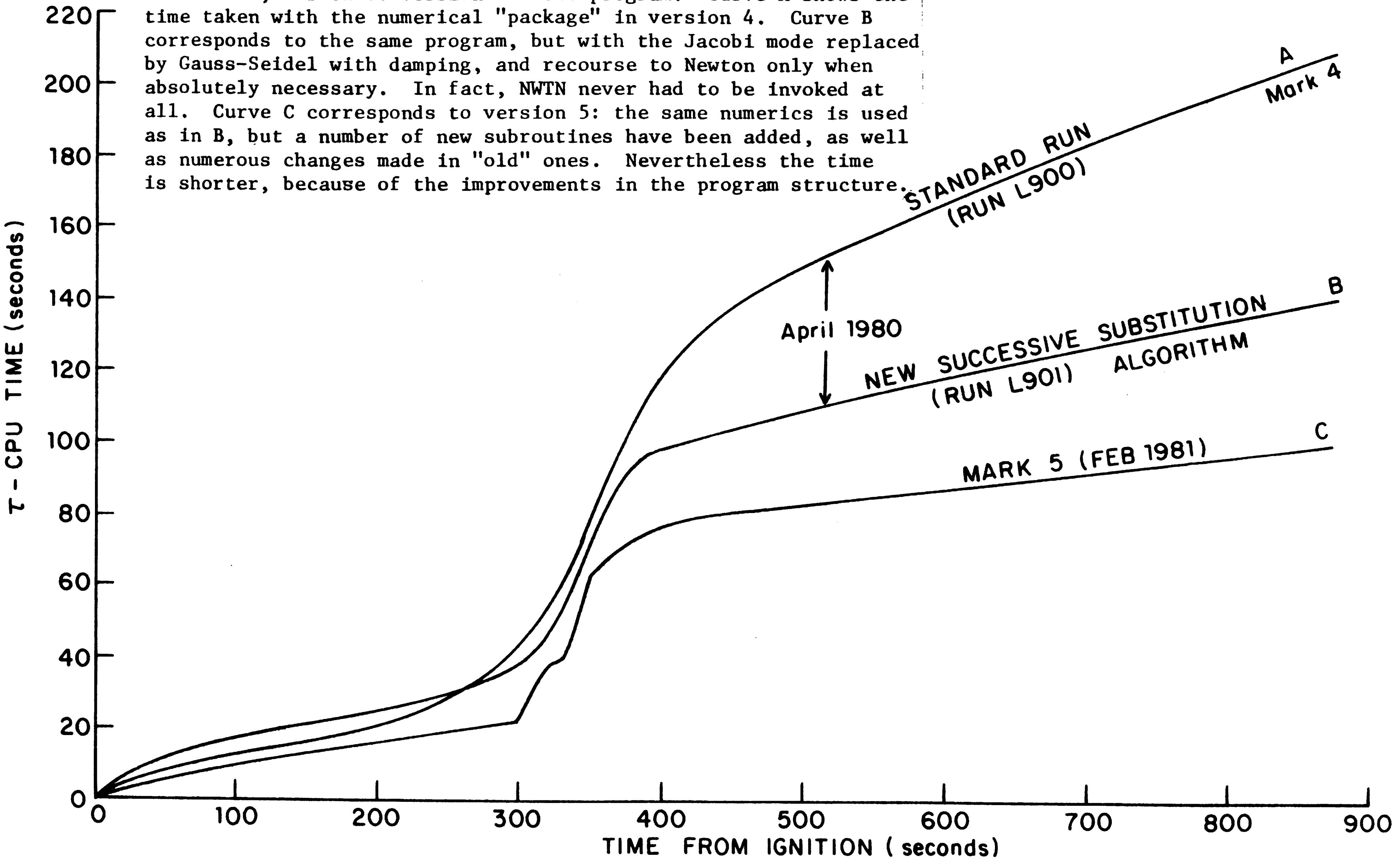


Fig. 6. Comparison of machine (CPU) time τ required for runs of different duration t, for three versions of the program. Curve A shows the time taken with the numerical "package" in version 4. Curve B corresponds to the same program, but with the Jacobi mode replaced by Gauss-Seidel with damping, and recourse to Newton only when absolutely necessary. In fact, NWTN never had to be invoked at all. Curve C corresponds to version 5: the same numerics is used as in B, but a number of new subroutines have been added, as well as numerous changes made in "old" ones. Nevertheless the time is shorter, because of the improvements in the program structure.

(damped Gauss-Seidel) algorithm, and with the cut-back in Δt whenever 35 G.S. iterations do not suffice for convergence, instead of the immediate switch to NWTN. For $t<260$, the time required is a little bit greater, but it is smaller thereafter; a 500-sec run takes about 25% less time than before, and for larger t, the decrease is closer to 30%. The principal reason for this decrease is that the present version of Gauss-Seidel is sufficiently robust, that the program never has to resort to using NWTN, in a standard run.

The new VENT routine (see sections II and V) was introduced thereafter; this made the program run a little slower: a 600-sec run with the old VNT routine(s) took about 98 sec--about 20 sec less than with run L901, whereas the new VENT took 109 sec. However, many changes were made in the program in order to produce Mark 5--changes which would generally be expected to slow down the calculation. Still, a 600-sec standard run now takes only about 87 sec, as seen from the curve marked Mark 5 in fig. 6. The increased speed is undoubtedly due to the improved logical flow. The amount of time needed for a Gauss-Seidel iteration cycle is approximately independent of the number of variables in ICOR; for the Standard Run, it is 20.6±1.0 ms/iteration. (For a 500-sec run, it averages 19.56 ms/iteration. The number of variables in ICOR varied between 24 and 45; the average for the run was about 39 or 40.) For a very similar run where object 2 was prevented from igniting, the maximum number of variables fell to 38, and the total time for the run indeed was shorter (9 seconds less); however, the mean time per iteration was <u>larger</u>: 20.46 ms! One Newton iteration, on the other hand, takes 1.0±.1 sec (it is proportional to N^3-- see the discussion on p. 16 of reference 18). When a more complex subroutine is used-- e.g. ABSRB3 instead of ABSRB2-- the calculation time goes up, of course: for a 400-sec run, the time per Gauss-Seidel iteration rose from 20.1 msec to 24.9 ms.

IV. The Program.

In this section we shall discuss the characteristics desirable in a program, and the degree to which we have achieved these goals with Mark 5. Following that we shall describe the organization and working of the program, largely in terms of structure and flow diagrams--first simple versions, then more detailed ones. Then the way the data is organized, placed in arrays, etc., is discussed. Any object, from the point of view of burning, exists in one of several qualitatively distinct "states". This categorization is useful to make in the program, so that calculations which are relevant in one state but not another, are carried out only in the former case. These are described. Next, the logical flow for the numerical algorithms is discussed. Then the COMMONS and their contents are given in detail; this may be thought of as the other half of the data structure. Finally, the input/output options available to the user are discussed.

We begin by considering what constitutes an ideal program, from our point of view:

1. Desiderata. There are a number of features which would be desirable in an ideal computer program. The principal ones, in order, might be:
   A. It should have universal applicability: no matter how small or large, how simple or complex the structure, a fire of any pre-assigned kind should be calculable. The fire could be in a mine shaft, an airplane flying at 35,000 feet, a furnace, or a house. The initial fire could be produced by a gas jet, a pool of spilled gasoline, a smoldering fire, etc. And on any material(s).
   B. It should be easy to use, so that no special sophistication is required from the user.

C. It should be highly reliable; that is, always stable and convergent, and accurate to any desired degree.

D. It should be flexible: we should be able to easily change any desired aspect--a parameter, a convergence criterion, the description of the physics of a subrpocess, etc. There should also be alternative modes of input and output.

E. It should be machine-independent and easily transportable.

F. It should be written in a language that's universal, or nearly so--this may be thought of as a corollary of item E, perhaps.

G. It should be transparent--that is, the logical flow should be clear and simple.

H. It should be modular: the program must be written in sections which are independent,so that a change made to one section should never mandate concomitant changes in any other section. In particular, it should be possible to program components of the physical model without any knowledge of the numerical solution procedure. Modularity is a way to accomplish items D and G, also.

Let us consider these in turn.

First, universality. At present, we are restricted to an enclosure which is a rectangular parallelepiped, with one to five rectangular vents in the walls, and one to five objects inside. The vents connect to the outside, which is assumed to be windless. We only consider flaming fires, but these can be a growing fire, a burner fire, and/or a pool fire. (Smoldering fires and wall fires are not yet incorporated.)

Second, ease of use. In order to run the program, the user merely types* "RUN MARK5" and then follows the instructions which appear on the

* This instruction may not be entirely machine-independent, however: different operating systems may have different conventions--the most likely being the demand for the three-letter "extension" signigying the compiled and linked program. Thus, for example, MARK5.EXE or MARK5.OUT might be required.

terminal screen (this will be described in more detail in section VIII). Input is requested from the user in simple formats, generally one item at a time. If an input error is made, there is generally at least one opportunity to correct it. Input can also be made in blocks, when the information is already there; thus, for example, if object #3 has the same physical characteristics as object #1 (say), that can all be input with two keystrokes, rather than a dozen. A card-input mode is also available. When all the required input has been supplied, the program automatically begins the computations.

Third, stability. The numerical algorithms we use still leave something to be desired: even though a wide class of problems will run through with no trouble, numerical instabilities still occur from time to time--perhaps 5-10% of the time.

Our experience so far has shown that most of the difficulties we've had arise from inadvertent introduction of discontinuities in the variables, in the "physical" subroutines. When successive substitution fails to give convergence, the NWTN algorithm is able to overcome the numerical difficulties some 80-90% of the time. Independent investigations (Ramsdell (27)) indicate that when there <u>is</u> a difficulty with NWTN, it often lies with the Jacobian matrix being ill-conditioned; this can be overcome (sometimes) by going to double precision. That option is not available in this version, however.

As indicated in the discussion in section III, we can generally expect accuracy within a few percent, with $\varepsilon_c = 3 \times 10^{-4}$ and $\Delta t_o = 2$ sec.

Flexibility. As discussed in the second item above, the interactive input is fairly flexible: it will forgive a wide class of errors and permit the user to change the input. We can, moreover, choose Δt as well as the length of the run, at run-time. In Mark 5, we <u>cannot</u> vary ε_c(TOLER), nor the initial value of the flame cone angle. But all the other parameters--geometric and physical--can be chosen at run time, should they differ from the default values.

The user may choose what kind of fire he wants, as well as having a few alternatives in the subroutines to be used. There are also two kinds of output mode , illustrated in Appendix C: first, a long-form output, where (nearly) every variable is displayed periodically (the period is user-chosen, with 10 seconds as the default value). Second, a short-form output, where only 8 variables are displayed; this is much more compact, taking only <u>one</u> line vs. 19+ lines per printout. The 8 variables may be chosen by the user at run time; if no special choice is made, the 8 variables which will be displayed are T_L $\equiv$ ZKLZZ (the layer temperature), T_s $\equiv$ ZKOZZ(2) (the surface temperature of object #2), T_w $\equiv$ ZKWZZ (the wall/ceiling temperature), h_L $\equiv$ ZHLZZ (the layer depth, or thickness), $y(O_2)$ $\equiv$ ZYLOZ (the mass concentration of oxygen), $\dot{m}_u$ $\equiv$ TMUZZ (the rate of hot gas mass outflow from the room), $-\dot{m}_f$ $\equiv$ -TMOZZ(1) (the rate of pyrolysis of the burning object), and R_f $\equiv$ ZRFZZ(1) (the radius of the initial fire). This is the "default" selection.

Machine independence and language. These two are really quite closely related. FORTRAN was chosen as the language to be used, because it has been in nearly universal use for scientific work, so that all interested institutions and probable end-users will have FORTRAN compilers. Moreover, other researchers can make contributions to this program <u>via</u> FORTRAN--written subroutines, and prepared numerical "packages" can also be used. Of course, different installations have somewhat different compilers and operating systems, and may use different versions of FORTRAN; we have written the program in as nearly machine-independent a form as was practicable within our time and resource constraints. Mark 5 is mostly in FORTRAN 77. As a result, the tapes have been successfully read and compiled in a wide variety of places: the National Bureau of Standards, the California Institute of Technology, in Sweden, in France, etc. The introduction and widespread use of the ANSI '77 FORTRAN has facilitated this task. A number of people have recommended the use of PASCAL or other structured language instead of FORTRAN, as it permits dynamic allocation of memory and data structures (such as records), easier modularization, and generally better data handling. ADA (pushed by the DOD) permits dynamic dimensioning, as well. However, neither of these languages at present have the near-universality of FORTRAN, and hence will not be adopted by us soon. (Not to speak of the onerous task of rewriting large programs in the new language!)

Transparency and modularity.

Ideally, a program should be written in "top-down" fashion. However, its authors are not programming experts, nor was such sophisticated expertise available at affordable cost. Hence this program grew, like many such programs, with a somewhat tangled structure. However, modularity was attempted from the first, with some success.

Mark 5 represents a significant step in the desired directions, as a result of substantial efforts made to simplify the structure of the program and improve modularity. There is no point in going into detailed description of all the changes that were made from Mark 4, though it is not amiss to mention a few of the present features: some 30-odd COMMON blocks have been concantenated into 13. The logical flow is indeed simpler and more "transparent"--i.e., easily grasped--than in Mark 4, and I shall shortly describe it. Modularization has been improved by making the argument list in the subroutines as short as possible* (frequently consisting of just one index); the needed variables are carried in COMMONs. This makes COMMON link all the subroutines, so that they are not, in fact, truly independent. The alternative would be to carry no COMMON, and have each subroutine (and its CALL) carry the whole list of variables involved. That has its own lack of appeal, of course.

We also have modularity in the numerics: the solution of the system of equations (6) is now found by any one of a set of numerical subroutines written for the purpose, at present including just JACB and NWTN (see Section III).

2. Program Structure.

I shall describe the organization of the program in two ways: first, the actual structure; that is, the placement and relationship of functional units and routines in relation to each other. This anatomical skeleton is vary useful for grasping the structure, and as an aid in following the flow charts which follow. This description is static, however, and only hints at function--the "physiology", to follow the organic analogy. The dynamics is therefore presented in flow diagrams.

The elementary structure is shown in fig. 7; for the more ambitious reader, it is shown in considerable detail in fig. 8. In the latter figure, the functional blocks indicated in fig. 7 are filled in with the appropriate subroutines and their interconnections. Fig. 9, on the other hand, gives the first--highly simplified--flow diagram, which describes what the program does. Fig. 10 gives a more detailed-- but

*Sub-subroutines sometimes may have a few more arguments.

still abbreviated--account of the program flows. Figs. 11-22 give the detailed flows for most of the program.

The most fundamental division of the program is into three parts: first, a set of subroutines which quantitatively describe all the relevant phenomena--i.e., the physics. Second, a set of subroutines which give one or more algorithms for the (numerical) solution of the resulting equations. Third, the controlling programs which direct the flow of the calculation.

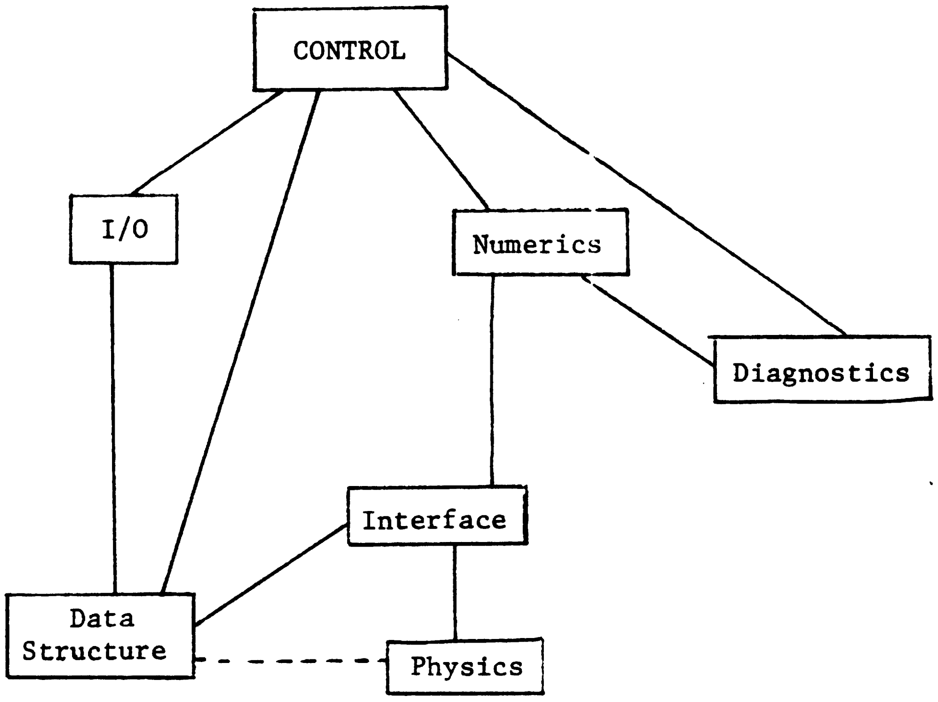


Fig. 7. Basic structure of the Fire Code. The directions of control are down along the indicated paths, except upon return. The dashed line between the bottom two boxes indicates that the subroutines which "do" the physics, also "know" something about the data structure, both from including COMMONs, and from (usually) having an array index in their calling argument.

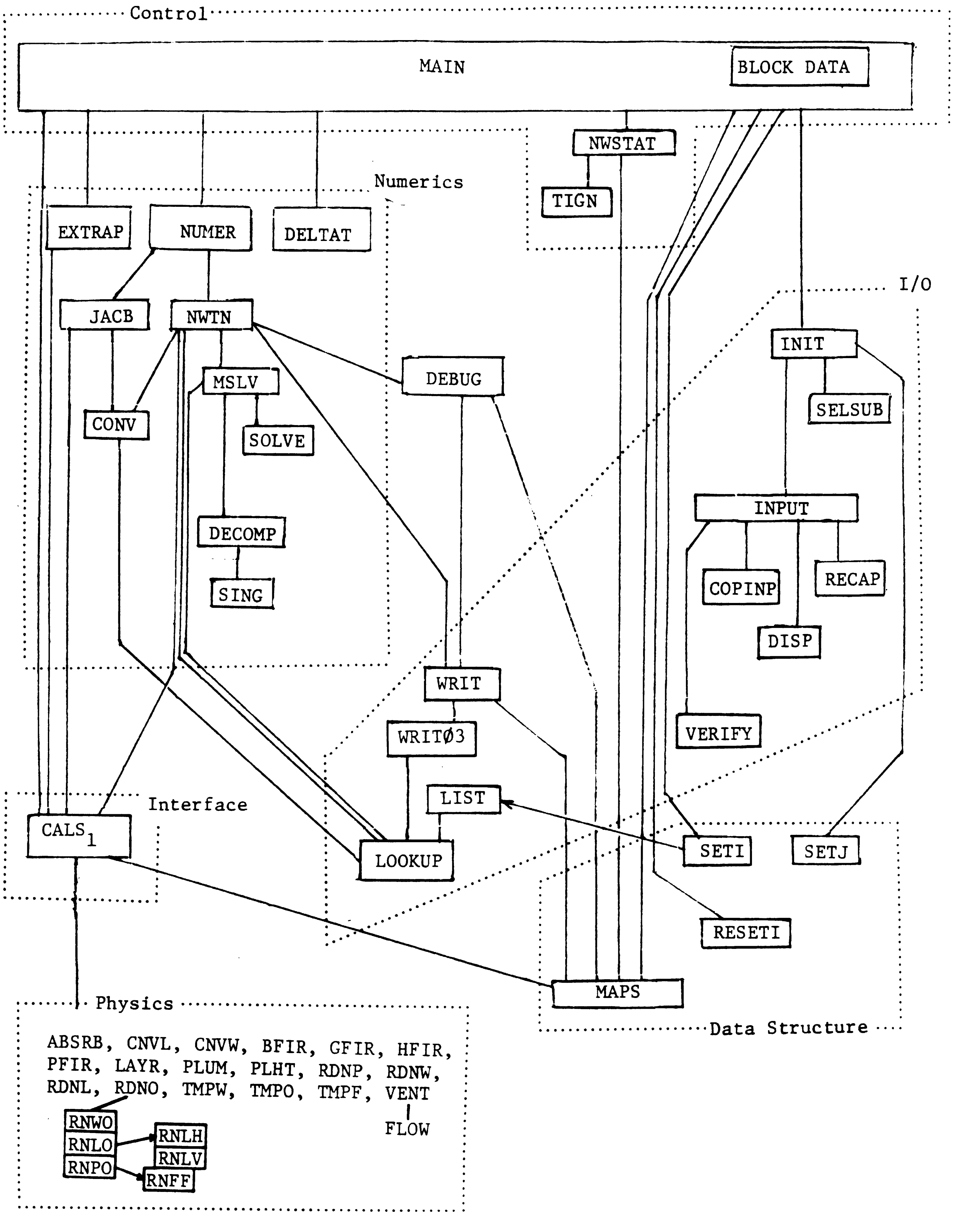


Fig. 8. Block diagram for the program. The functional boxes shown in fig. 7 have here been replaced by their constituent subroutines. This may be compared with fig. 1 of CFC III. Also see Appendix 6.2 at the end of the section on COMMONS in Chapter IV.

At a just slightly more detailed level, we may note <u>six</u> functions; first: numerical, physical, and controlling, just as described in the previous paragraph. Besides these, an input/output (I/O) section, another which establishes the data structure, and (finally) a set of interface programs which mediate between the physics and the numerics. (There is also a "diagnostic" subroutine, DEBUG, which escapes categorization.) See fig. 7. The casual reader may now wish to skip to fig. 9, and then to figs. 18a and b.

Except for ABSRB, all the physical subroutine names are four-letter combinations. Every subroutine name is of course meant to be suggestive of its function. For the physical subroutines, the shortness of the names makes recognition difficult; they are described in detail in section V, however, so that should pose no problem. For a few of the others, the meaning may be made clearer from the following short table:

Table I. Partial dictionary of subroutine names:

CALS	=	call subroutines
COPINP	=	copy input
DISP	=	display
MSLV	=	matrix solver
NWSTAT	=	new state(s)
SELSUB	=	select subroutine
SING	=	singular matrix detector
TIGN	=	time of ignition
WRITØ3	=	short-form output

By convention each subroutine name represents a class of more-or-less interchangeable subroutines which perform the same logical role in the program operation. For example, CFC5 contains ABSRB1 and ABSRB2 (one or the other alternative is used in a calculation); they are both referred to by the generic name ABSRB.

Before we can discuss the logical flow intelligently, we must understand something about the naming and storage conventions for the variables. Thus, we have:

3. <u>Data Structure</u>.

Within the physical subroutines, variables appear with names which are alphanumeric labels whose meaning is given by the conventions detailed in Appendix A, and listed in Appendix B. In the higher-level subroutines, however, it is greatly advantageous to be able to work with an abstract array X instead of a collection of individual names. This enables us to perform operations on the whole vector of physical variables at once by means of Fortran DO loops.

The necessary correspondence between distinct names at one level and abstract arrays at another is achieved by writing the different lists of symbols in common block* /VAR/ in different ways: Within each of the physical subroutines, /VAR/ appears with a list of all the physical variables in a prescribed order. This is the current list:

```
   COMMON /VAR/                  TELZR(5),TELZD(5),ZMLZZ(5),TMLZZ(5),
                                 ZELZZ(5),TELZZ(5),ZHLZZ(5),ZKLZZ(5),
                                 ZYLOZ(5),ZYLOZ(5),ZYLMZ(5),ZYLSZ(5),
                                 ZYLWZ(5),ZPRZZ(5),ZULZZ(5),RMFILL(25),
                                 FQLOR(5),FQWOR(5),FQPOR(5),ZKOZZ(5),
                                 ZMOZZ(5),TMOZZ(5),TEOZZ(5),ZHPZZ(5),
                                 TMPZZ(5),TEPZZ(5),TMPLU(5),TEPZR(5),
                                 ZRFZZ(5),TPSI(5),OBFILL(130),
                                 TEUZZ(5),TMUZZ(5),TMDZZ(5),
                                 VTFILL(85),
                                 FQLWR(5,2),FQPWR(5,2),FQLWD(5,2),
                                 ZKWZZ(5,2),WLFILL(30,2)
```

Within each of the higher-level subroutines, /VAR/ appears with the vector XIN occupying the same storage locations:

```
      COMMON /VAR/ XIN(500)
```

We currently allow a maximum of 100 variables. These may be distributed in various ways, however. Thus we might (in principle, at least) have five rooms--corresponding, according to the discussion in section III, to 5 x 15 = 75 variables--one object, one vent, and one wall. Or, we might have <u>one</u> room, five objects, one vent and one wall. Or one room, four objects, five vents, and one wall. Etc. With an eye to possible expansion, then, we make room for 100 variables indexed by ROOM, 200 for those indexed by OBJECT, 100 for those indexed by VENT, and 100 for those indexed by WALL. This yields the (quite sparse) array XIN, and its image in commons VAR and OLDVAR. See the discussion for MAPS and for common /VAR/.

The sparse array XIN is then mapped onto the (relatively densely packed) array X <u>via</u> the subroutine MAPS. Then the numerical subroutines take X

*VAR is shorthand for "variable", here.

(and, if necessary, the previous timestep values XØ and XØØ) to find the first iteration value XK.

Although the organization of COMMONS is part of the data structure, they are important enough to warrant their own subsection--see 6.

In the physical subroutines, no variable ever appears as an abstract array element. In the control program MAIN and the numerical programs no variable ever appears in the fire code notation. The same is now true of CALS, and CALS1, the physical-subroutine-calling programs, although in Mark III and IV CALS was an interface in which both forms of reference appeared.

Next, consider the logical flow. Start with fig. 9. The master program which directs the flow of the calculation to the various subroutines is called MAIN, and is shown in fig. 11 below.

There are a number of auxiliary control and interface subroutines: EXTRAP, DELTAT, NUMER, RESETI, NWSTAT, TIGN, SELSUB, SETI, SETJ, INIT, MAPS, INPUT3, DEBUG, WRIT, CALS, and CALS1, of which the first five are called by MAIN, and are schematized below, in figs. 12-16. TIGN is called by NWSTAT, and is shown in fig. 17. Their functions are as follows:

EXTRAP provides the initial "guess" for the values x_i at a new time $t + \Delta t$; that is, it provides a predictor, and the numerics then supply the corrector. The extrapolation is (essentially) quadratic for the Gauss-Seidel mode (requiring, therefore, two previous timestep values, as well as the current one), and linear in the NWTN mode (experience has shown that we do rather better with linear extrapolation, in the latter case).

DELTAT calculates Δt for the next timestep. For example, if we have been moving along with Δt_0-sized steps, and an ignition occurs, then we interpolate all values between t and $t + \Delta t_0$ to the value at $t + \delta t$, the moment of ignition; the next timestep must then be $\Delta t = \Delta t_0 - \delta t$. Again, we (generally) display results at multiples of 10 sec; if $t < 10m$ while $t + \Delta t_0 > 10m$, again we must choose $\Delta t < \Delta t_0$, such that $t + \Delta t = 10m$. And so forth.

NUMER. Subroutine JACB is fast and good in the early stages of a calculation; however, it will sometimes fail. Subroutine NWTN is more powerful, but ponderously slow. For the package to be reliable and efficient the control must be flexible enough to pass from one of these methods to another without disrupting the computation. This switching is handled by the control

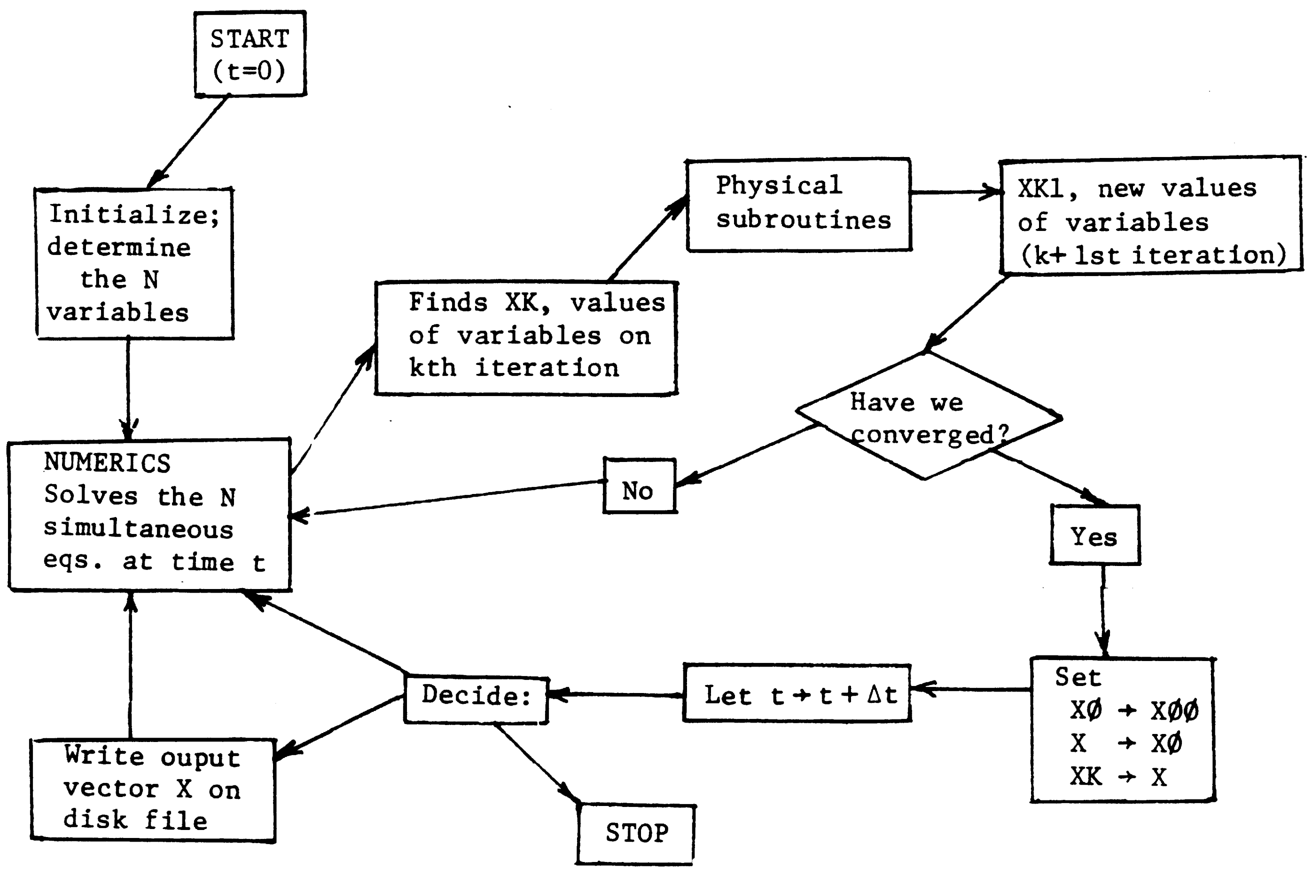

START (t=0)
Initialize; determine the N variables
NUMERICS Solves the N simultaneous eqs. at time t
Finds XK, values of variables on kth iteration
Physical subroutines
XK1, new values of variables (k+ 1st iteration)
Have we converged?
No
Yes
Set XØ → XØØ X → XØ XK → X
Let t → t + Δt
Decide:
Write ouput vector X on disk file
STOP

Fig. 9. Highly simplified schematic (flow diagram) of the entire program. Note that XK is the array of values of the (N) variables in the current iteration cycle. XK1 is the set of values in the next iteration, resulting from passing XK through the physical subroutines (successive substitution case--see text. The more complex case is not illustrated here; see Fig. 18b, instead). When we have converged, XK is placed in X; the previous set of values in the array X is placed in XØ, corresponding to the "previous timestep", and (similarly) XØ goes into XØØ. NUMERICS does all the fancy work of extrapolating to get predictors, finding appropriate mean values, interpolating, finding Jacobian matrices, etc.

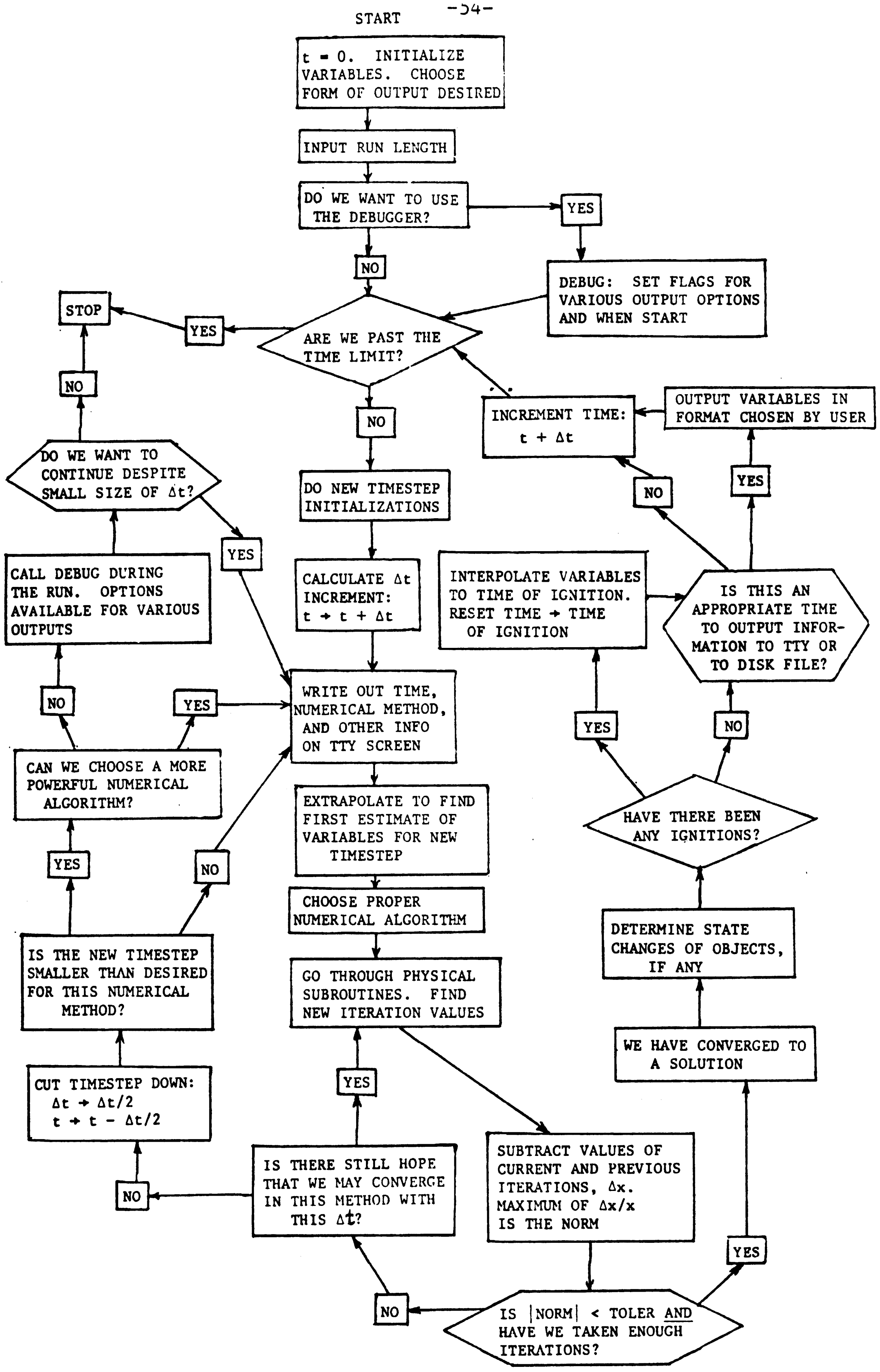

START
t = 0. INITIALIZE VARIABLES. CHOOSE FORM OF OUTPUT DESIRED
INPUT RUN LENGTH
DO WE WANT TO USE THE DEBUGGER?
YES
DEBUG: SET FLAGS FOR VARIOUS OUTPUT OPTIONS AND WHEN START
NO
ARE WE PAST THE TIME LIMIT?
YES
STOP
NO
NO
DO WE WANT TO CONTINUE DESPITE SMALL SIZE OF Δt?
YES
CALL DEBUG DURING THE RUN. OPTIONS AVAILABLE FOR VARIOUS OUTPUTS
NO
YES
CAN WE CHOOSE A MORE POWERFUL NUMERICAL ALGORITHM?
YES
NO
IS THE NEW TIMESTEP SMALLER THAN DESIRED FOR THIS NUMERICAL METHOD?
CUT TIMESTEP DOWN: Δt → Δt/2 t → t - Δt/2
NO
DO NEW TIMESTEP INITIALIZATIONS
CALCULATE Δt INCREMENT: t → t + Δt
WRITE OUT TIME, NUMERICAL METHOD, AND OTHER INFO ON TTY SCREEN
EXTRAPOLATE TO FIND FIRST ESTIMATE OF VARIABLES FOR NEW TIMESTEP
CHOOSE PROPER NUMERICAL ALGORITHM
GO THROUGH PHYSICAL SUBROUTINES. FIND NEW ITERATION VALUES
YES
IS THERE STILL HOPE THAT WE MAY CONVERGE IN THIS METHOD WITH THIS Δt?
NO
SUBTRACT VALUES OF CURRENT AND PREVIOUS ITERATIONS, Δx. MAXIMUM OF Δx/x IS THE NORM
IS |NORM| < TOLER AND HAVE WE TAKEN ENOUGH ITERATIONS?
NO
YES
WE HAVE CONVERGED TO A SOLUTION
DETERMINE STATE CHANGES OF OBJECTS, IF ANY
HAVE THERE BEEN ANY IGNITIONS?
YES
NO
INTERPOLATE VARIABLES TO TIME OF IGNITION. RESET TIME → TIME OF IGNITION
IS THIS AN APPROPRIATE TIME TO OUTPUT INFORMATION TO TTY OR TO DISK FILE?
YES
NO
OUTPUT VARIABLES IN FORMAT CHOSEN BY USER
INCREMENT TIME: t + Δt

Fig. 10. Short version of the entire program. This is, effectively, an expanded version of fig. 9 (also see Appendix D of ref. 28).

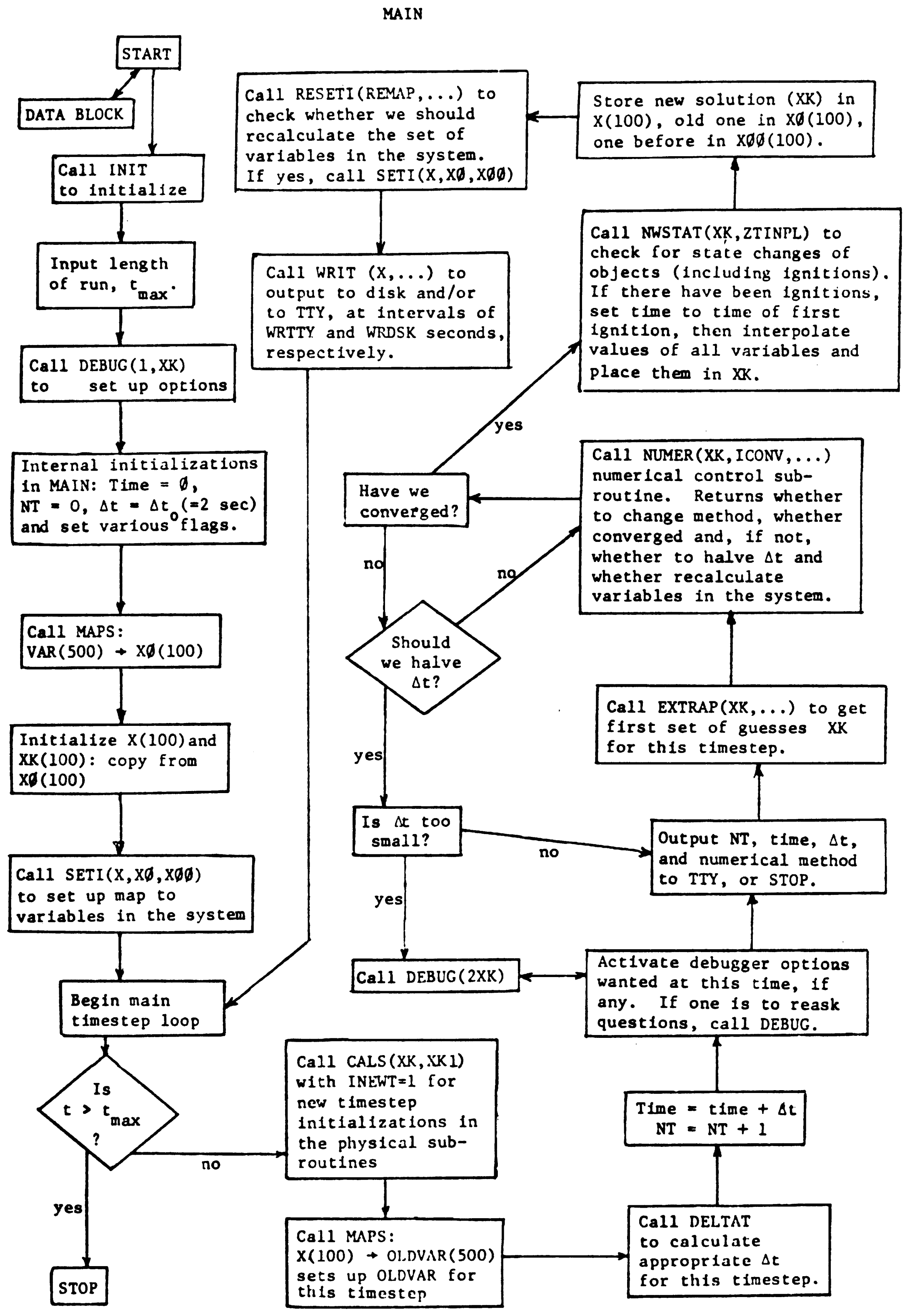

MAIN
START
DATA BLOCK
Call INIT to initialize
Input length of run, t_{max}.
Call DEBUG(1,XK) to set up options
Internal initializations in MAIN: Time = Ø, NT = 0, Δt = Δt_0(=2 sec) and set various flags.
Call MAPS: VAR(500) → XØ(100)
Initialize X(100) and XK(100): copy from XØ(100)
Call SETI(X,XØ,XØØ) to set up map to variables in the system
Begin main timestep loop
Is t > t_{max} ?
yes
STOP
no
Call CALS(XK,XK1) with INEWT=1 for new timestep initializations in the physical subroutines
Call MAPS: X(100) → OLDVAR(500) sets up OLDVAR for this timestep
Call DELTAT to calculate appropriate Δt for this timestep.
Time = time + Δt NT = NT + 1
Activate debugger options wanted at this time, if any. If one is to reask questions, call DEBUG.
Call DEBUG(2XK)
Output NT, time, Δt, and numerical method to TTY, or STOP.
Call EXTRAP(XK,...) to get first set of guesses XK for this timestep.
Call NUMER(XK,ICONV,...) numerical control subroutine. Returns whether to change method, whether converged and, if not, whether to halve Δt and whether recalculate variables in the system.
Have we converged?
yes
no
Should we halve Δt?
no
yes
Is Δt too small?
no
yes
Call NWSTAT(XK,ZTINPL) to check for state changes of objects (including ignitions). If there have been ignitions, set time to time of first ignition, then interpolate values of all variables and place them in XK.
Store new solution (XK) in X(100), old one in XØ(100), one before in XØØ(100).
Call RESETI(REMAP,...) to check whether we should recalculate the set of variables in the system. If yes, call SETI(X,XØ,XØØ)
Call WRIT (X,...) to output to disk and/or to TTY, at intervals of WRTTY and WRDSK seconds, respectively.

Fig. 11. Flow diagram for MAIN, the principal flow-controlling program.

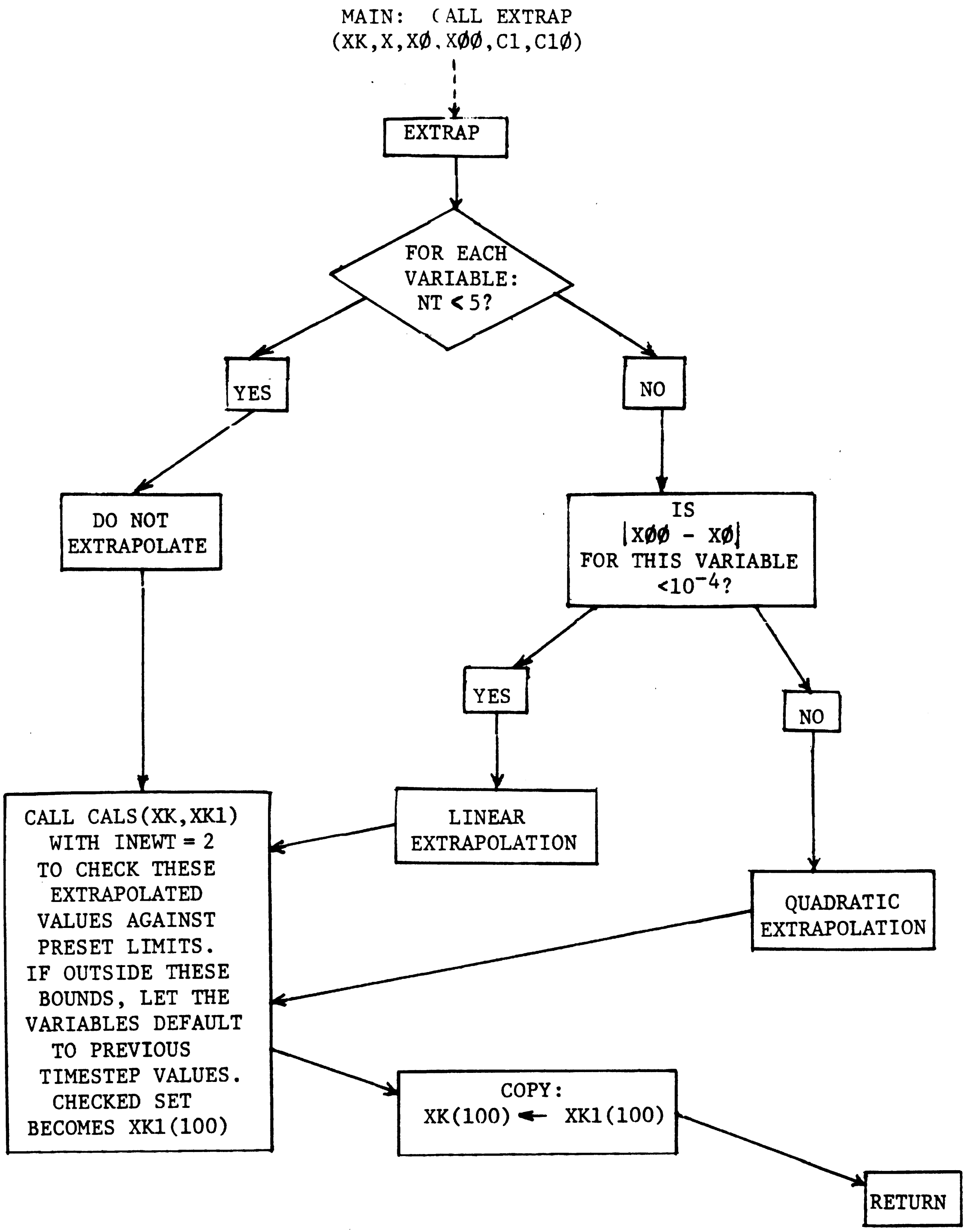

MAIN: CALL EXTRAP (XK,X,XØ,XØØ,C1,C1Ø)
EXTRAP
FOR EACH VARIABLE: NT < 5?
YES
NO
DO NOT EXTRAPOLATE
IS |XØØ - XØ| FOR THIS VARIABLE <10⁻⁴?
YES
NO
CALL CALS(XK,XK1) WITH INEWT = 2 TO CHECK THESE EXTRAPOLATED VALUES AGAINST PRESET LIMITS. IF OUTSIDE THESE BOUNDS, LET THE VARIABLES DEFAULT TO PREVIOUS TIMESTEP VALUES. CHECKED SET BECOMES XK1(100)
LINEAR EXTRAPOLATION
QUADRATIC EXTRAPOLATION
COPY: XK(100) ← XK1(100)
RETURN

Fig. 12. Flow diagram for EXTRAP, numerical subroutine auxiliary to MAIN. It finds whether to use zeroth, first, or second-order extrapolation in initial estimate of variables at a new timestep, and then does it.

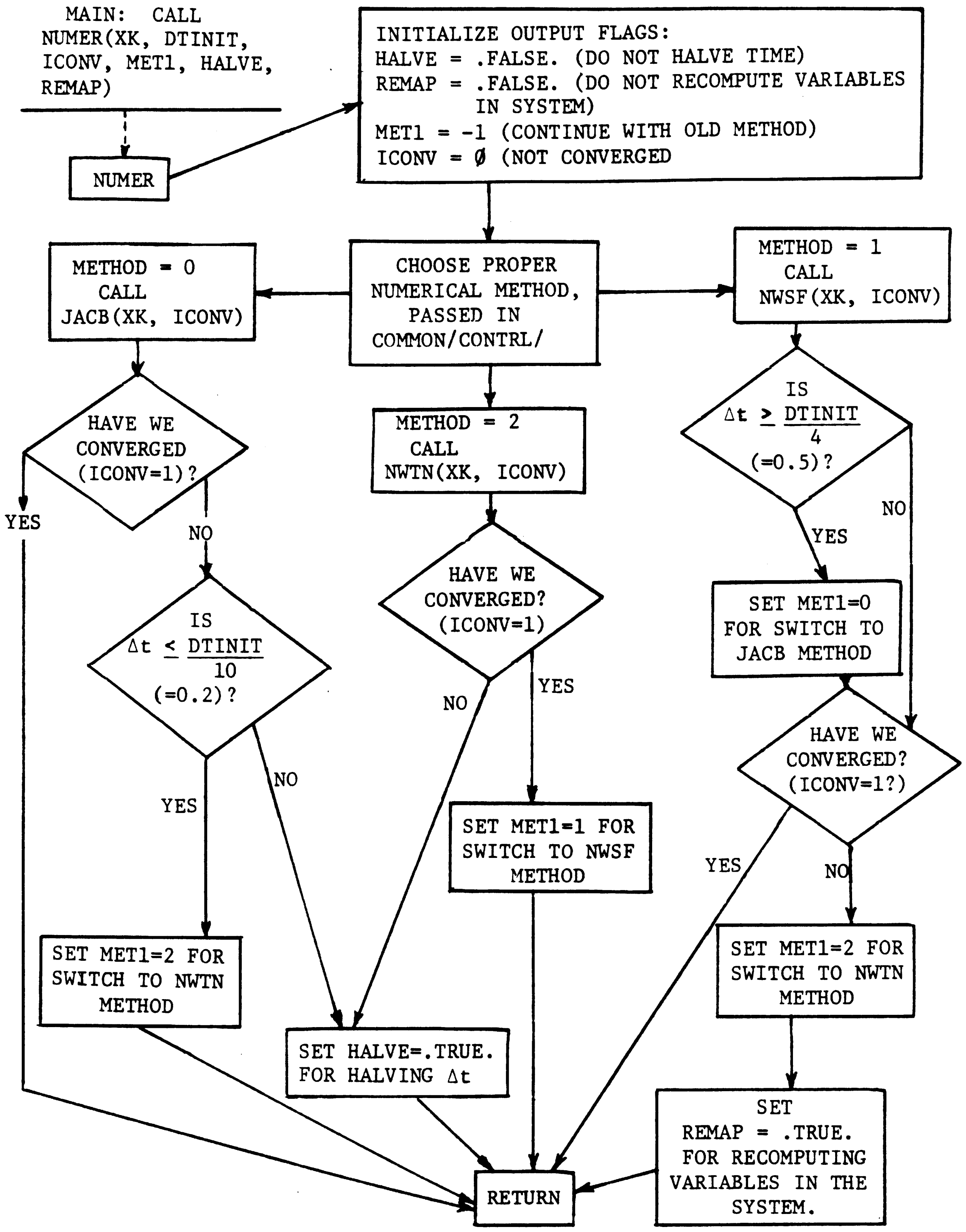

MAIN: CALL NUMER(XK, DTINIT, ICONV, MET1, HALVE, REMAP)
NUMER
INITIALIZE OUTPUT FLAGS:
HALVE = .FALSE. (DO NOT HALVE TIME)
REMAP = .FALSE. (DO NOT RECOMPUTE VARIABLES IN SYSTEM)
MET1 = -1 (CONTINUE WITH OLD METHOD)
ICONV = Ø (NOT CONVERGED
METHOD = 0 CALL JACB(XK, ICONV)
CHOOSE PROPER NUMERICAL METHOD, PASSED IN COMMON/CONTRL/
METHOD = 1 CALL NWSF(XK, ICONV)
HAVE WE CONVERGED (ICONV=1)?
YES
NO
METHOD = 2 CALL NWTN(XK, ICONV)
IS Δt ≥ DTINIT/4 (=0.5)?
NO
YES
IS Δt ≤ DTINIT/10 (=0.2)?
YES
NO
HAVE WE CONVERGED? (ICONV=1)
NO
YES
SET MET1=0 FOR SWITCH TO JACB METHOD
HAVE WE CONVERGED? (ICONV=1?)
SET MET1=1 FOR SWITCH TO NWSF METHOD
YES
NO
SET MET1=2 FOR SWITCH TO NWTN METHOD
SET HALVE=.TRUE. FOR HALVING Δt
SET MET1=2 FOR SWITCH TO NWTN METHOD
SET REMAP = .TRUE. FOR RECOMPUTING VARIABLES IN THE SYSTEM.
RETURN

Fig. 13. Flow diagram for NUMER, subroutine auxiliary to MAIN. NUMER decides which numerical method to use.

subroutine NUMER (called by MAIN). It determines which numerical procedure to use next; it sets a number of flags as well, which will subsequently tell the program whether or not to cut Δt in two and whether or not to reset ICOR. Both NWTN and JACB have ICONV as one of their calling arguments; ICONV is a flag which is set to 1 on exit if a converged solution has been found and to Ø otherwise. The flow between NWTN and JACB is detailed in section III.

SELSUB. The user may wish to choose a physical subroutine different from those in the default package (when alternatives are available). He does this at run-time,and the information is organized into the array IVRSN(20), in SELSUB; this is subsequently used in CALS (see section 6: COMMONS, for a description of IVRSN).

Finally, RESETI determines whether the set of variables "in the system" (that is, those belonging in ICOR) should be recalculated at any time. In fact, ICOR should be recomputed if (a) a physical quantity changes from being apparently a constant, to a variable. Or vice-versa, a variable ceases to vary, and appears to become constant. Or a variable becomes so small as to be physically insignificant (in which case it ought no longer to be carried along). Or, if a physically negligible variable grows to a meaningful size. A quantity will be said to be "constant" if its value changes by less than its VMIN value, from one timestep to the next. Similarly, it will be considered insignificant if its value is less than its VMIN value.

We also recompute ICOR when (b) DTSCAL seconds have passed since the last recomputation, and (c) on the first 10 timesteps of the program.

4. Physical States.

Consider one of the combustible objects in the room. At some time into the fire it may in fact ignite, either through autoignition, piloted ignition, or contact. From that moment forward, it is no longer necessary to continue calculating its surface temperature: it will stay at (essentially) a constant temperature (roughly T_{ig}). On the other hand, we must calculate its burning rate thereafter, the fireplume it develops, and so forth. The simplest way to carry this kind of information in the program is to associate an index with each object, ISTAT(KO), which specifies what state it's in. These states are listed in Table II.

Table II. The states an object can be in.*

When ISTAT is	Object is
1	cold (not involved)
2	heating, but not pyrolyzing
3	pyrolyzing, but not burning
4	smoldering
5	flaming
5a	a growing fire
5b	a pool fire or ignited fire
5c	a gas burner
6	---
7	---
8	burning charcoal
9	extinguished
10	consumed--i.e., burned out

This brings us to subroutines TIGN and NWSTAT. TIGN (called by NWSTAT) introduces the criteria for ignition used here: attainment of the autoignition temperature T_{ig}, or physical contact with a flame. When ignition to flaming occurs, ISTAT for that object is changed (from 1-4 or 9) to 5 (5b for autoignition, 5a for contact ignition), and the time at which ignition occurred, TCHNG, is calculated by a simple linear interpolation. The other state change we have included is burnout. This is here defined as having the (burning) mass fall to 1 mg or less. When this happens, the state of the object is changed from ISTAT=5 to 10, and the moment of burnout found by interpolation. TIGN is schematized in fig. 17.

Subroutine NWSTAT (called by MAIN) orders temporally the state changes (if any) occurring in a calculated interval, prints out the information, and finds the values of all the physical variables at the earliest of the

* This list deviates a little from the one on p. 7 of reference 28, and supersedes it (in the program, this list is found in TIGN). In the present version (Mark 5), only the indices 1, 2, 5, and 10 are used. The subcategories under "flaming fire" are flagged by the index KFTYP (for "flame type"), which takes on the values 1, 2, and 3, corresponding to 5a, b, and c, respectively.

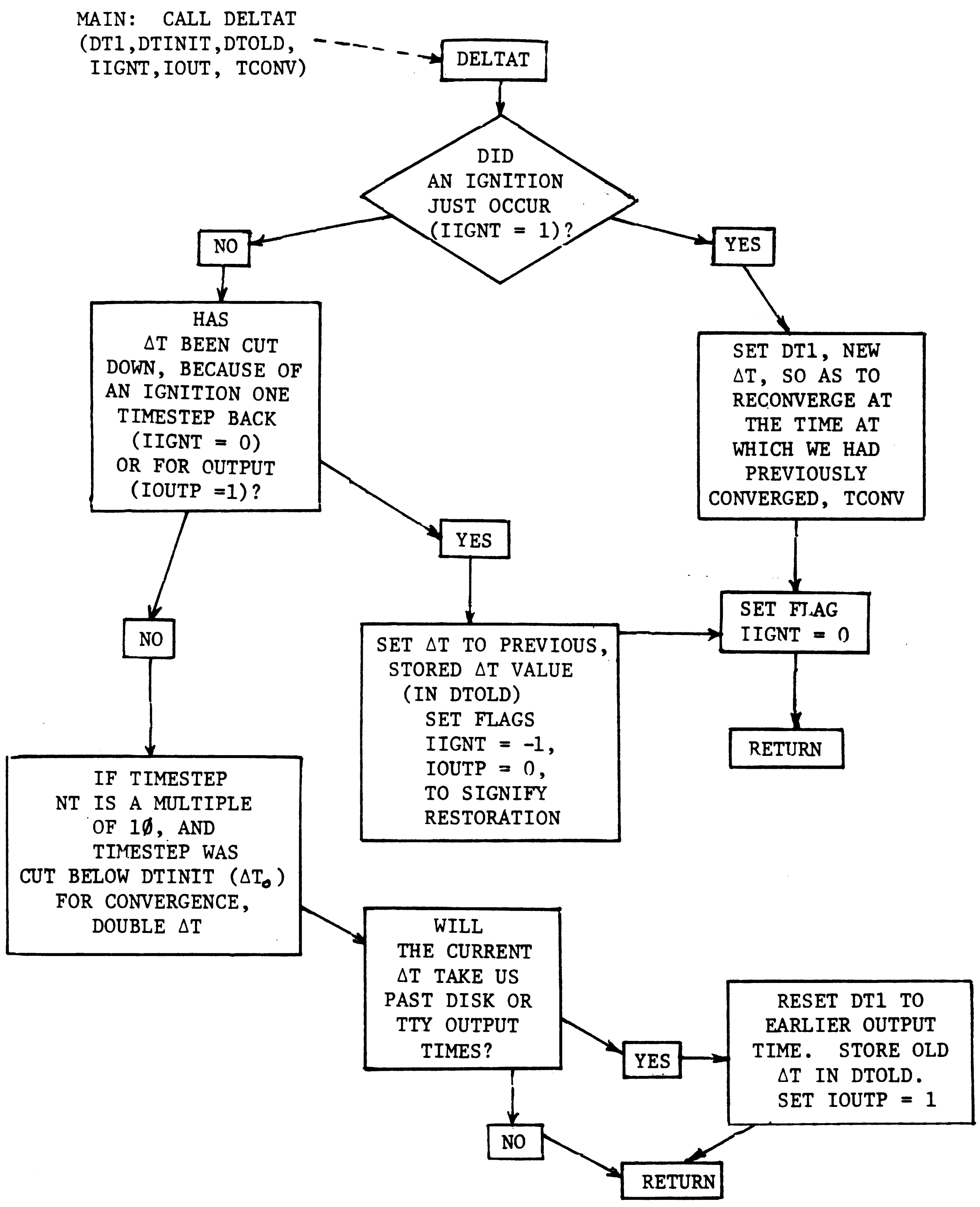

MAIN: CALL DELTAT (DT1,DTINIT,DTOLD, IIGNT,IOUT, TCONV)
DELTAT
DID AN IGNITION JUST OCCUR (IIGNT = 1)?
NO
YES
HAS ΔT BEEN CUT DOWN, BECAUSE OF AN IGNITION ONE TIMESTEP BACK (IIGNT = 0) OR FOR OUTPUT (IOUTP =1)?
SET DT1, NEW ΔT, SO AS TO RECONVERGE AT THE TIME AT WHICH WE HAD PREVIOUSLY CONVERGED, TCONV
YES
NO
SET ΔT TO PREVIOUS, STORED ΔT VALUE (IN DTOLD) SET FLAGS IIGNT = -1, IOUTP = 0, TO SIGNIFY RESTORATION
SET FLAG IIGNT = 0
RETURN
IF TIMESTEP NT IS A MULTIPLE OF 1∅, AND TIMESTEP WAS CUT BELOW DTINIT (ΔT∅) FOR CONVERGENCE, DOUBLE ΔT
WILL THE CURRENT ΔT TAKE US PAST DISK OR TTY OUTPUT TIMES?
YES
RESET DT1 TO EARLIER OUTPUT TIME. STORE OLD ΔT IN DTOLD. SET IOUTP = 1
NO
RETURN

Fig. 14. Flow diagram for DELTAT, numerical subroutine auxiliary to MAIN. It finds what value of Δt to use at any step.

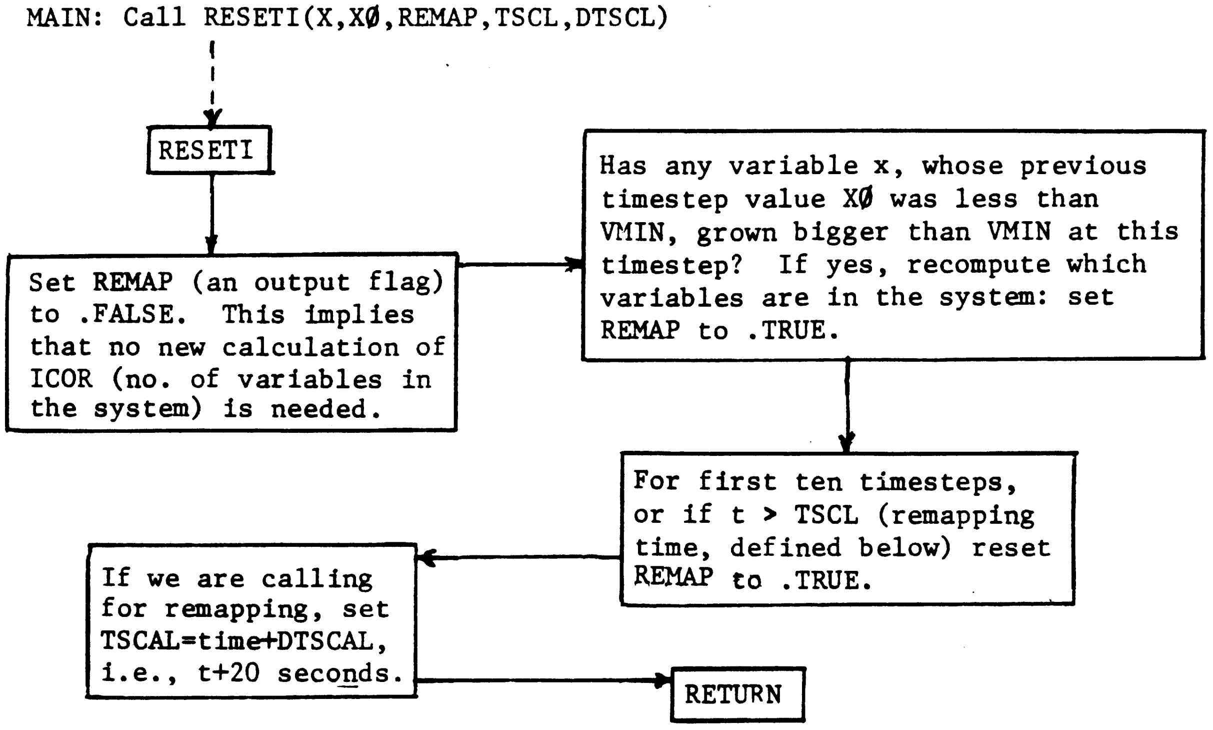
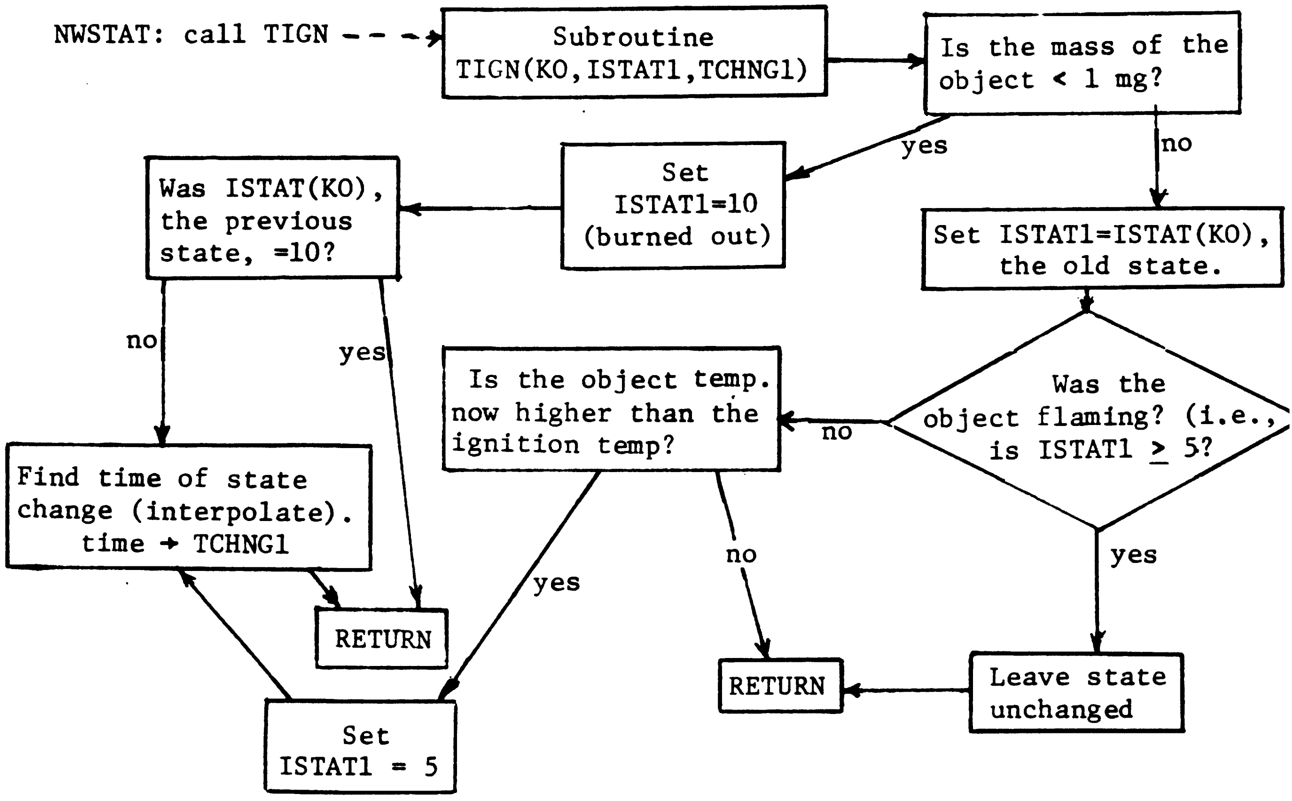

MAIN: Call RESETI(X,XØ,REMAP,TSCL,DTSCL)
RESETI
Set REMAP (an output flag) to .FALSE. This implies that no new calculation of ICOR (no. of variables in the system) is needed.
Has any variable x, whose previous timestep value XØ was less than VMIN, grown bigger than VMIN at this timestep? If yes, recompute which variables are in the system: set REMAP to .TRUE.
For first ten timesteps, or if t > TSCL (remapping time, defined below) reset REMAP to .TRUE.
If we are calling for remapping, set TSCAL=time+DTSCAL, i.e., t+20 seconds.
RETURN

Fig. 15. Flow diagram for RESETI, subroutine auxiliary to MAIN.

Fig. 17. Flow diagram for TIGN, control subroutine auxiliary to MAIN.


NWSTAT: call TIGN
Subroutine TIGN(KO,ISTAT1,TCHNG1)
Is the mass of the object < 1 mg?
yes
no
Set ISTAT1=10 (burned out)
Set ISTAT1=ISTAT(KO), the old state.
Was ISTAT(KO), the previous state, =10?
no
yes
Was the object flaming? (i.e., is ISTAT1 ≥ 5?
no
yes
Is the object temp. now higher than the ignition temp?
yes
no
Find time of state change (interpolate). time → TCHNG1
RETURN
RETURN
Leave state unchanged
Set ISTAT1 = 5

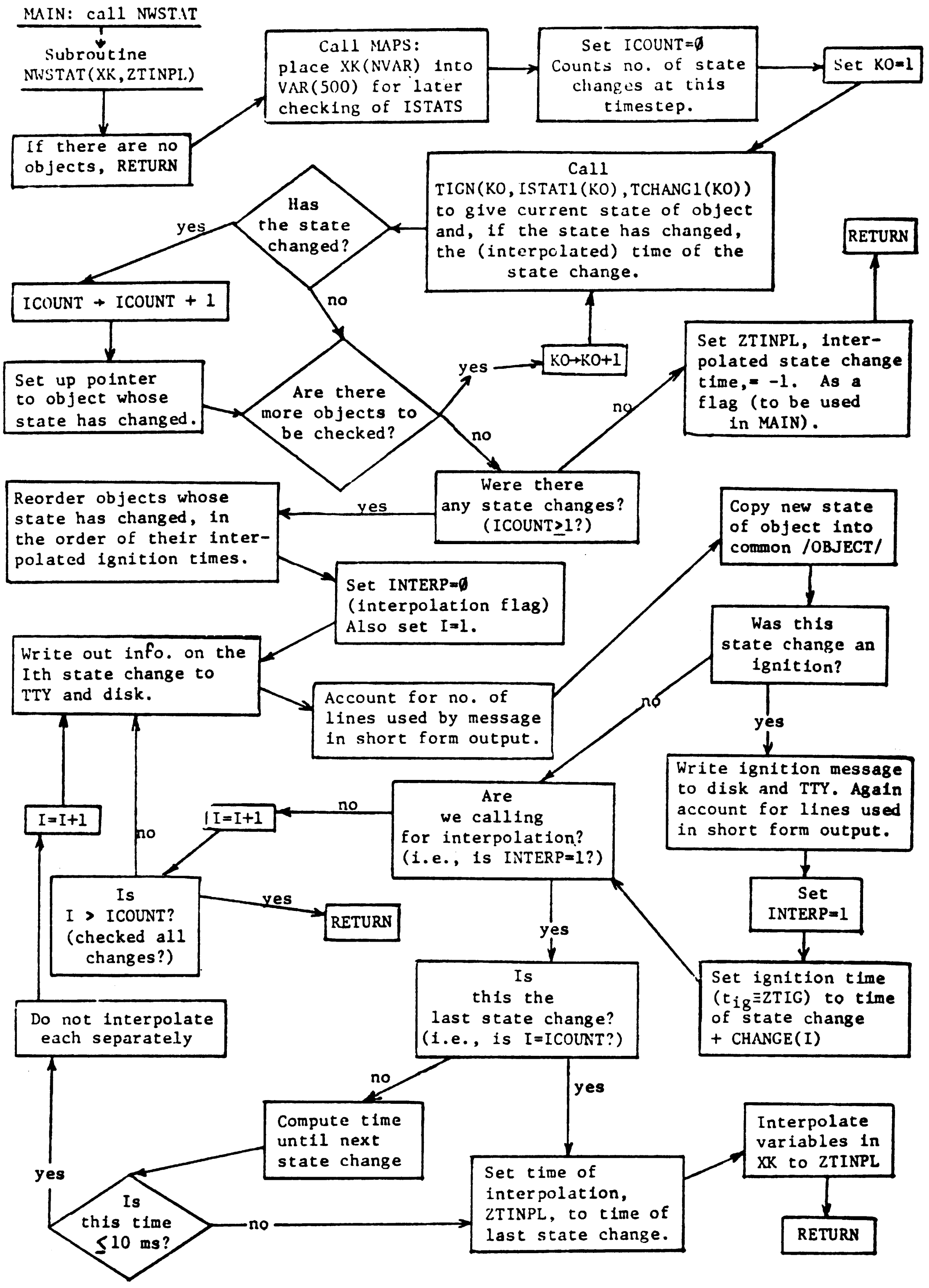

MAIN: call NWSTAT
Subroutine NWSTAT(XK,ZTINPL)
If there are no objects, RETURN
Call MAPS: place XK(NVAR) into VAR(500) for later checking of ISTATS
Set ICOUNT=Ø Counts no. of state changes at this timestep.
Set KO=1
Call TIGN(KO,ISTAT1(KO),TCHANG1(KO)) to give current state of object and, if the state has changed, the (interpolated) time of the state change.
Has the state changed?
yes
no
ICOUNT → ICOUNT + 1
Set up pointer to object whose state has changed.
Are there more objects to be checked?
yes
KO→KO+1
no
RETURN
Set ZTINPL, interpolated state change time, = -1. As a flag (to be used in MAIN).
no
Were there any state changes? (ICOUNT≥1?)
yes
Reorder objects whose state has changed, in the order of their interpolated ignition times.
Set INTERP=Ø (interpolation flag) Also set I=1.
Write out info. on the Ith state change to TTY and disk.
Account for no. of lines used by message in short form output.
Copy new state of object into common /OBJECT/
Was this state change an ignition?
no
yes
Write ignition message to disk and TTY. Again account for lines used in short form output.
Set INTERP=1
Set ignition time (t_ig≡ZTIG) to time of state change + CHANGE(I)
Are we calling for interpolation? (i.e., is INTERP=1?)
no
I=I+1
I=I+1
no
Is I > ICOUNT? (checked all changes?)
yes
RETURN
yes
Is this the last state change? (i.e., is I=ICOUNT?)
no
yes
Do not interpolate each separately
Compute time until next state change
yes
Is this time ≤10 ms?
no
Set time of interpolation, ZTINPL, to time of last state change.
Interpolate variables in XK to ZTINPL
RETURN

Fig. 16. Flow diagram for NWSTAT, control subroutine auxiliary to MAIN. This subroutine, called by MAIN, finds whether the state of any object has changed in a given timestep; if this has happened, the nature of the change is noted, appropriate flags set, and the time at which this happened, found.

state-change time(s) by linear interpolation. As discussed earlier, we then recalculate the solution at time $t+\Delta t$ with the object in the altered state, by starting with the interpolated values at $t+\delta t$, and taking a timestep of size $\delta t'$ such that $\delta t+\delta t=\Delta t$. NWSTAT is schematized in fig. 16.

CALS (called by MAIN, EXTRAP, JACB, and NWTN) is a simple controlling routine which calls the various physical subroutines. It does so in a pre-arranged order (which can be changed by changing the DATA statement for the array IORDER in CALS). CALS starts with the (linear) array of variables XK, and finds the next iteration values XK1 (this notation was of course suggested by $\vec{x}^k$ and $\vec{x}^{k+1}$).

5. Numerical algorithms:There are N variables involved in a given run ($N \leq 100$). A block of space of size N, called JCOR, is then set aside in memory. N is called NVAR in the program. Some of these variables may not vary at all, but be constant; some others may be zero or (effectively) vanishingly small. There is no point in trying to get these to "converge"; moreover, computation time increases not linearly with the number of variables n in the numerical system, but in some places in proportion to n^2 or n^3 (when NWTN is involved). As a result it is highly important that the number of variables in the active system be kept to a minimum. Hence the above-mentioned variables are excluded from the Newton calculation, as well. The remaining subset of (significant) variables is placed in ICOR. ICOR shows which position in the system corresponds to which variable stored in the complete sequence of physical variables in common block /VAR/: for example, if ICOR(1)=1 and ICOR(2)=4, then variable 1 and 4 are in the numerical system but variables 2 and 3 have been excluded. Now suppose we have one vent and two objects in the room. Then from eq. (1), N=54. Suppose that ICOR has 40 variables in it, at some point in the calculation. The numerical procedures involved in the successive-substitution and Newton modes are schematized in figs. 18a and 18b.

The flow diagrams for the numerical subroutines are given in greater detail in figs. 20 and 21. Then figs. 22-24 outline the interface routines SETI, SETJ, AND MAPS. Subroutine SETJ (called by INIT) is a generalized JCOR-initializing routine. SETI (called by MAIN) sets up ICOR--that is, the set of variables "in the system"--those to be used in the Jacobian and to be checked for convergence. Note that JCOR and ICOR appear in COMMON /MAP/. Various point and arrays are set up by SETI and SETJ.

Subroutine MAPS (called by MAIN, DEBUG, WRIT, and CALS) takes a sparse

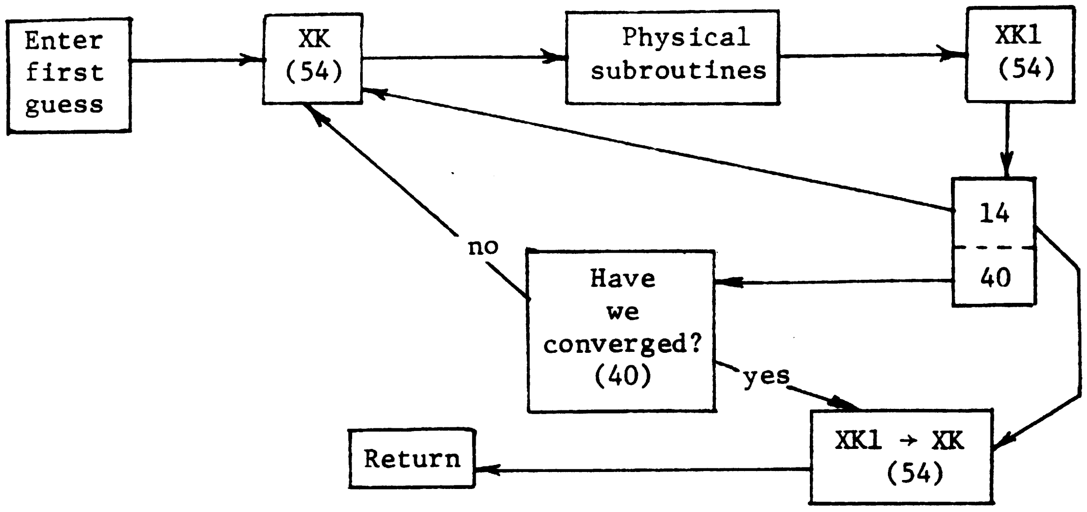


Fig. 18a. Schematic flow diagram for the numerical procedures in the successive substitution mode(s). The numbers in parentheses represent the number of variables involved (see the text).

array of variables such as XIN or VAR and maps it onto a more concentrated (and more economical) array, such as X. Generally, XIN→X and XOLD→XØ. It also does the inverse, depending on the index ICTRL with which it is called. These mappings are achieved via the array JCOR. See fig. 19.

6. COMMONS and other arrays.

This section is based on one written by Barry London. There is some overlap between the descriptions given here, and the previous text. However, some redundancy is probably a virtue, in so complex a program . Note that this is also part of the data structure.

In CFC V, much of the communication between subroutines is done using common blocks. Here, we will give a brief description of the structure and function of each of these common blocks, and list the variables found in each.

1) VAR:

Common VAR (for "variable") contains the set of physical variables calculated by the program. This common has enough space for 500 variables: the first 100 spaces are reserved for room variables, the next 200 spaces for object variables, the next 100 spaces for vent variables, and the last 100 spaces for wall variables. Most variables are indexed to room (KR), object (KO), vent (KV), or wall (KW). See section III.1 for details. Hence they must be DIMENSIONED to the maximum number that can be input; these maxima are MR, MO, MV, and MW, respectively. These are all (still) given the arbitrary value 5 (in BLOCK DATA, following MAIN in the program). MR=5, however, is

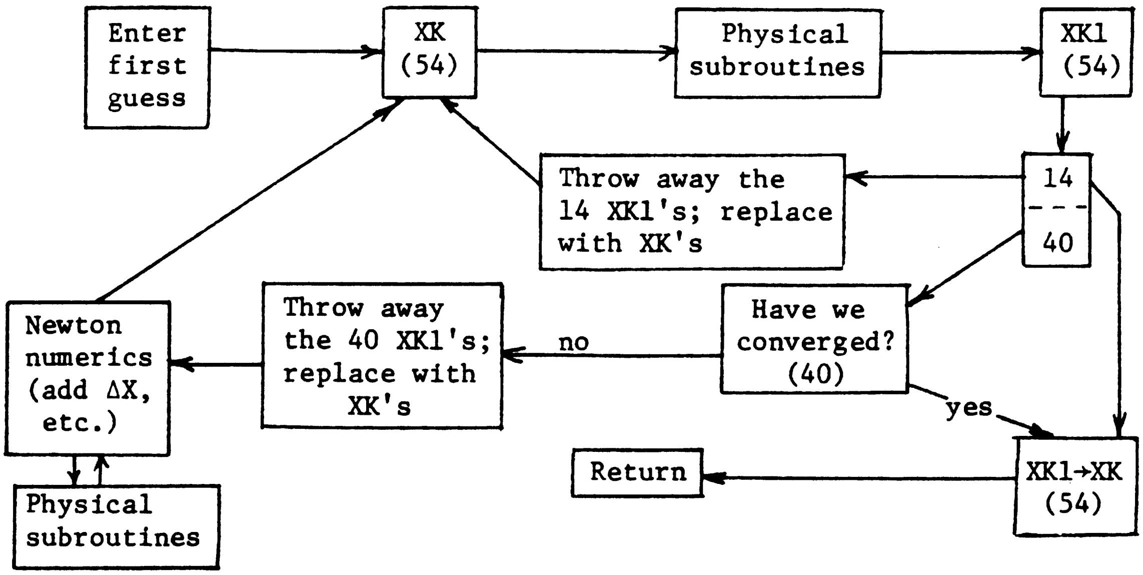


Fig. 18b. Schematic flow diagram for NWTN mode. Note that much of it is similar to 18a.

a fiction: as a practical matter, we can only run one room at a time, with Mark 5. To run more than one room would require considerable programming modifications. Walls are dimensioned to (MW,2) to account for their two sides. There are currently 15 room variables, 14 object variables, 3 vent variables, and 4 x 2 = 8 wall variables (see section III.1). The remaining space at the end of each section is filled by the array RMFILL(25), OBFILL(130), VTFILL(85), and WLFILL(30,2), respectively.

The starting address of each section mentioned above is given in the array OFFSET(4), in subroutine SETJ and also in LOOKUP. The number of each type of variable is given in the array INVAR(4), also in subroutine SETJ.

The higher level control subroutines deal with all of the physical subrouti as a block. For these cases, all variables of VAR are referenced by the array XIN(500).

The variables of VAR are all initialized in the subroutine INIT. The commo VAR is present as explicitly named variables in INIT, the long form output routine WRIT, and in all of the physical subroutines. It is present as XIN(500) in the numerical and interface subroutines.

Common VAR is a working common: it is used for several purposes during each iteration (e.g. computation of each column of the Jacobian matrix, checking for convergence, and output, if desired). VAR may or may not change during an

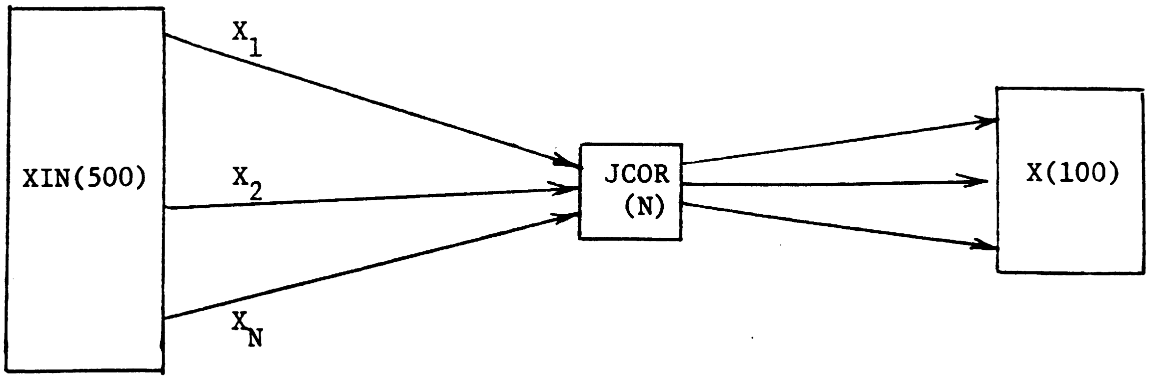


Fig. 19. Schematic of MAPS. The mapping is shown from XIN to X (corresponding to ICTRL=0), but it is in fact a one-to-one mapping between any of the large, sparse arrays, and any of the more compact arrays; hence the arrows can also go in the other direction (for ICTRL=1). Note that XIN is isomorphic with VAR, and that N is shorthand for NVAR.

iteration cycle; see appendix 6.2 at the end of this subsection. Previous iteration values are stored in the array XK(100), in MAIN.

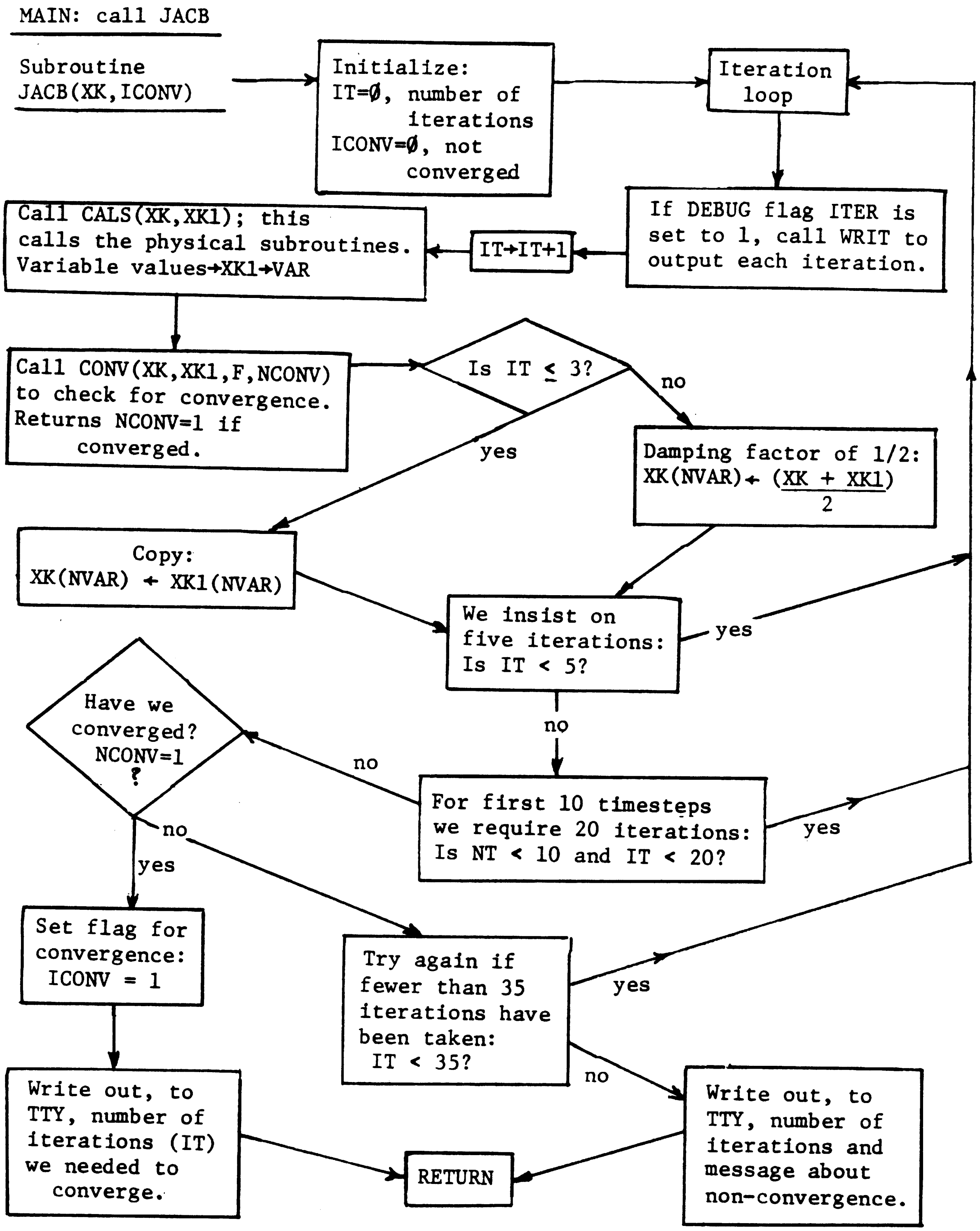
The following is a list of variables as they appear in CFC V, in commons VAR, OLDVAR, NEWVAR, VAR1, NEWVAR1, and in the long form output. The order in this list corresponds to reading the output table in rows. However, the two items preceded by an asterisk, do not appear on the output list. The meaning of each symbol is given in the Dictionary, in Appendix B.

TELZR(KR), TELZD(KR), ZMLZZ(KR), TMLZZ(KR), ZELZZ(KR), TELZZ(KR), ZHLZZ(KR), ZKLZZ(KR), ZYLOZ(KR), ZYLDZ(KR), ZYLMZ(KR), ZYLSZ(KR), ZYLWZ(KR), ZPRZZ(KR), *ZULZZ(KR), RMFILL(25); FQLOR(KO), FQWOR(KO), FQPOR(KO), ZKOZZ(KO), ZMOZZ(KO), TMOZZ(KO), TEOZZ(KO), ZHPZZ(KO), TMPZZ(KO), TEPZZ(KO), *TMPLU(KO), TEPZR(KO), ZRFZZ(KO), TPSI(KO), OBFILL(130); TEUZZ(KV), TMUZZ(KV), TMDZZ(KV), VTFILL(85); FQLWR(KW,J), FQPWR(KW,J), FQLWD(KW,J), ZKWZZ(KW,J), WLFILL(30,2). RMFILL, OBFILL, VTFILL, and WLFILL are (room, object, vent, wall) filler "variables" to expand the room,... section of common VAR to 100, 200, 100, and 100 words, respectively.

2) OLDVAR:

This common gives the (converged) values of those physical variables which were computed and stored in common VAR at the previous timestep. When present as explicitly named variables, the variable names are identical to the names in VAR, except that they end with the letter 'P' (for "previous"). Otherwise, it is present as the array XINOLD(500).

This common is reset once each timestep by MAIN, and remains unchanged during the entire timestep. The use and distribution of the two forms of common



MAIN: call JACB
Subroutine JACB(XK,ICONV)
Initialize: IT=∅, number of iterations ICONV=∅, not converged
Iteration loop
If DEBUG flag ITER is set to 1, call WRIT to output each iteration.
IT←IT+1
Call CALS(XK,XK1); this calls the physical subroutines. Variable values→XK1→VAR
Call CONV(XK,XK1,F,NCONV) to check for convergence. Returns NCONV=1 if converged.
Is IT ≤ 3?
no
yes
Damping factor of 1/2: XK(NVAR)← (XK + XK1)/2
Copy: XK(NVAR) ← XK1(NVAR)
We insist on five iterations: Is IT < 5?
yes
no
Have we converged? NCONV=1 ?
no
For first 10 timesteps we require 20 iterations: Is NT < 10 and IT < 20?
yes
no
yes
Set flag for convergence: ICONV = 1
Try again if fewer than 35 iterations have been taken: IT < 35?
yes
no
Write out, to TTY, number of iterations (IT) we needed to converge.
RETURN
Write out, to TTY, number of iterations and message about non-convergence.

Fig. 20. Flow diagram for the numerical subroutine JACB. It implements the Gauss-Seidel version of successive substitution; given the array XK, it finds XK1.

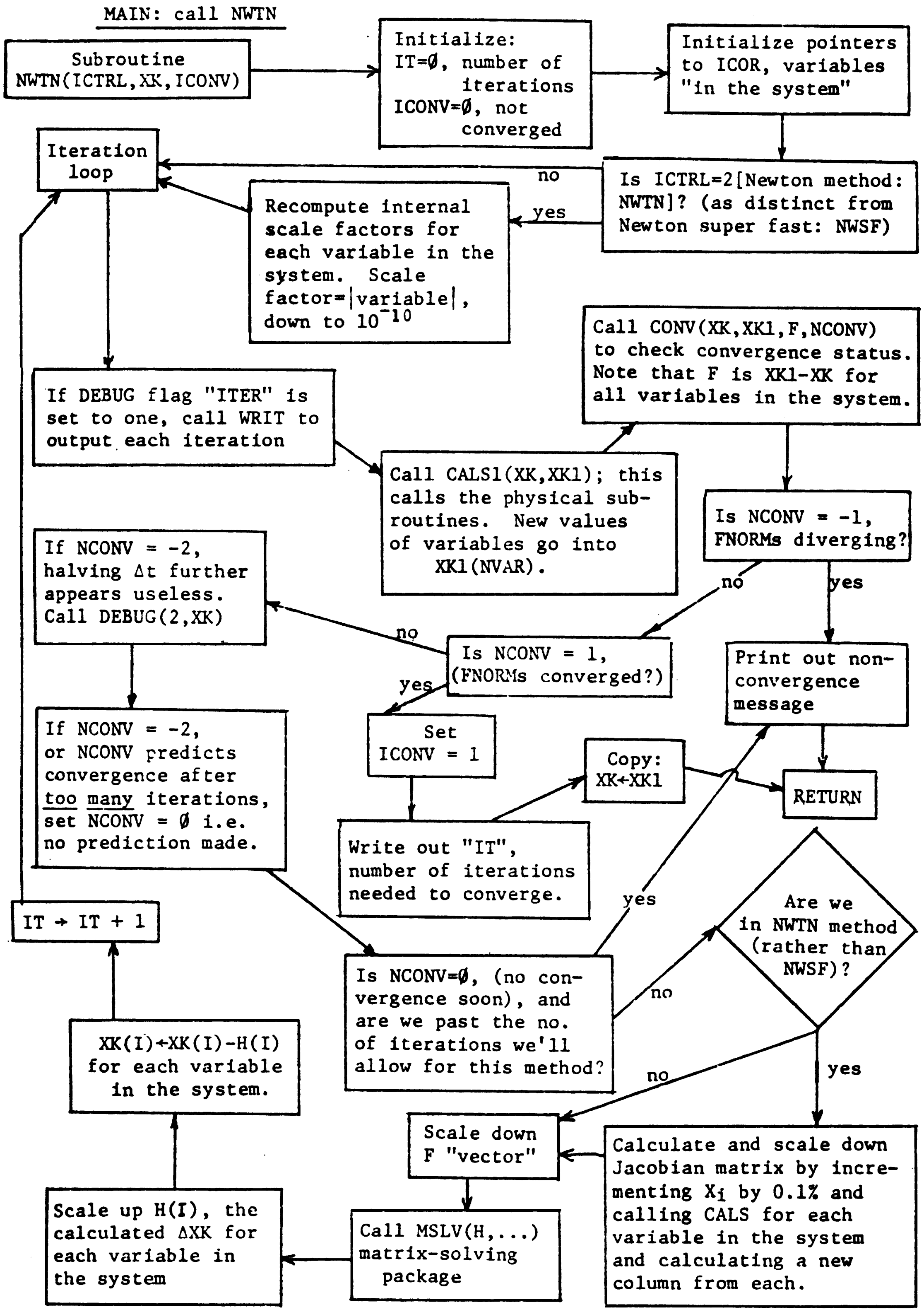

MAIN: call NWTN
Subroutine NWTN(ICTRL,XK,ICONV)
Initialize: IT=Ø, number of iterations ICONV=Ø, not converged
Initialize pointers to ICOR, variables "in the system"
Iteration loop
no
Is ICTRL=2[Newton method: NWTN]? (as distinct from Newton super fast: NWSF)
yes
Recompute internal scale factors for each variable in the system. Scale factor=|variable|, down to 10^-10
Call CONV(XK,XK1,F,NCONV) to check convergence status. Note that F is XK1-XK for all variables in the system.
If DEBUG flag "ITER" is set to one, call WRIT to output each iteration
Call CALS1(XK,XK1); this calls the physical subroutines. New values of variables go into XK1(NVAR).
Is NCONV = -1, FNORMs diverging?
no
yes
If NCONV = -2, halving Δt further appears useless. Call DEBUG(2,XK)
no
Is NCONV = 1, (FNORMs converged?)
yes
Print out non-convergence message
Set ICONV = 1
Copy: XK←XK1
RETURN
If NCONV = -2, or NCONV predicts convergence after too many iterations, set NCONV = Ø i.e. no prediction made.
Write out "IT", number of iterations needed to converge.
yes
Are we in NWTN method (rather than NWSF)?
IT → IT + 1
Is NCONV=Ø, (no convergence soon), and are we past the no. of iterations we'll allow for this method?
no
no
yes
XK(I)←XK(I)-H(I) for each variable in the system.
Scale down F "vector"
Calculate and scale down Jacobian matrix by incrementing Xi by 0.1% and calling CALS for each variable in the system and calculating a new column from each.
Scale up H(I), the calculated ΔXK for each variable in the system
Call MSLV(H,...) matrix-solving package

Fig. 21. Flow diagram for the numerical subroutines NWTN and NWSF. Given the array XK, they find the array XK1 using a multivariate version of Newton-Raphson.

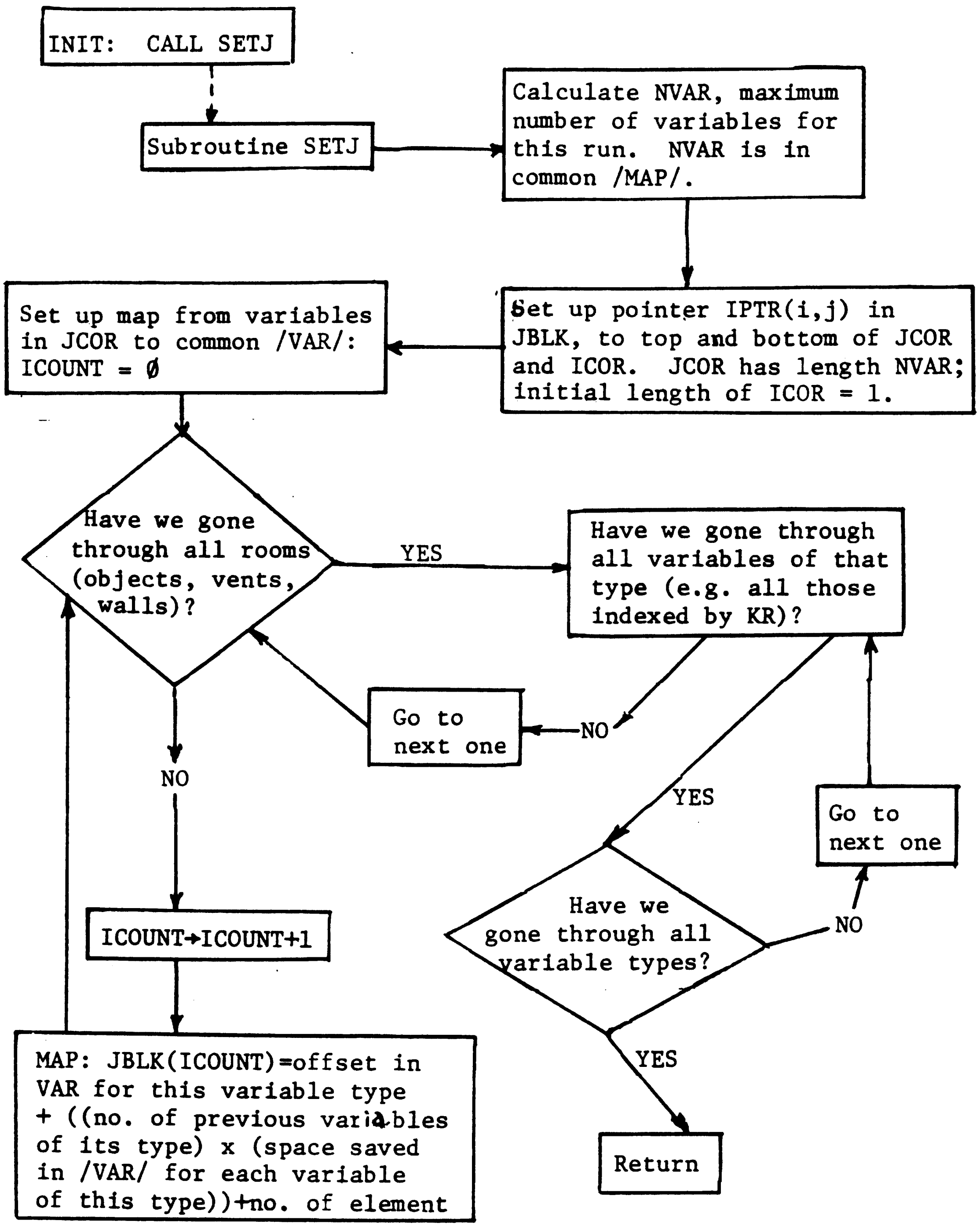

INIT: CALL SETJ
Subroutine SETJ
Calculate NVAR, maximum number of variables for this run. NVAR is in common /MAP/.
Set up pointer IPTR(i,j) in JBLK, to top and bottom of JCOR and ICOR. JCOR has length NVAR; initial length of ICOR = 1.
Set up map from variables in JCOR to common /VAR/: ICOUNT = Ø
Have we gone through all rooms (objects, vents, walls)?
YES
Have we gone through all variables of that type (e.g. all those indexed by KR)?
Go to next one
NO
NO
YES
Go to next one
ICOUNT→ICOUNT+1
Have we gone through all variable types?
NO
YES
MAP: JBLK(ICOUNT)=offset in VAR for this variable type + ((no. of previous variables of its type) x (space saved in /VAR/ for each variable of this type))+no. of element
Return

Fig. 22. Flow diagram for the interface (data structure) subroutine SETJ. This subroutine finds the maximum number of variables needed in a run, then sets up JCOR and maps it into VAR.

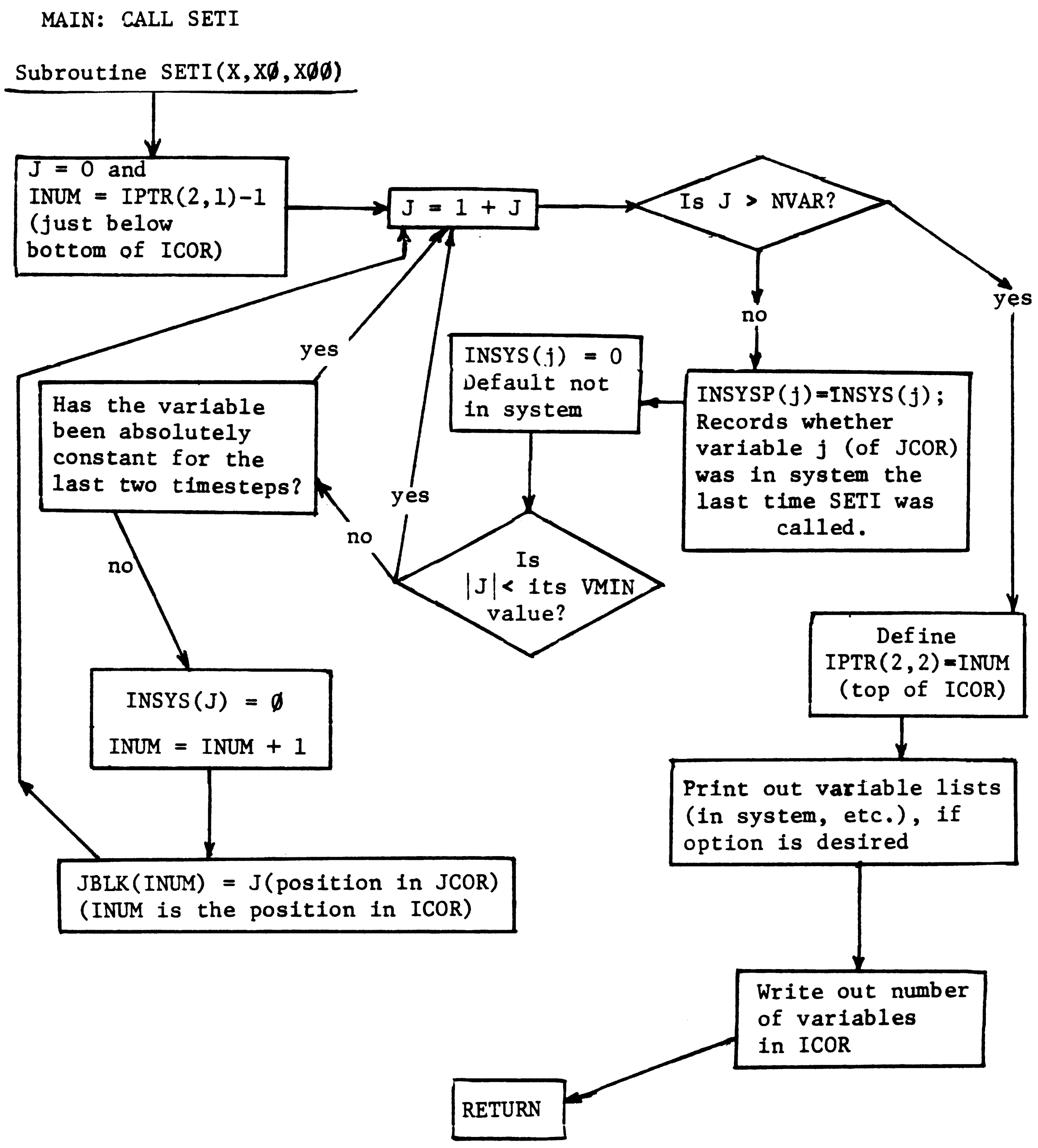

MAIN: CALL SETI
Subroutine SETI(X,XØ,XØØ)
J = 0 and INUM = IPTR(2,1)-1 (just below bottom of ICOR)
J = 1 + J
Is J > NVAR?
no
yes
INSYSP(j)=INSYS(j); Records whether variable j (of JCOR) was in system the last time SETI was called.
INSYS(j) = 0 Default not in system
Is |J| < its VMIN value?
yes
no
Has the variable been absolutely constant for the last two timesteps?
yes
no
INSYS(J) = Ø
INUM = INUM + 1
JBLK(INUM) = J(position in JCOR) (INUM is the position in ICOR)
Define IPTR(2,2)=INUM (top of ICOR)
Print out variable lists (in system, etc.), if option is desired
Write out number of variables in ICOR
RETURN

Fig. 23. Flow diagram for the interface subroutine SETI (part of data structure). This subroutine sets up ICOR, and maps it into JCOR and JBLK.

OLDVAR is otherwise identical to that of the two forms of common VAR.

3) NEWVAR:

NWTN uses common NEWVAR to compute the Jacobian matrix without disturbing VAR. When present as explicitly named variables, the variable names are identical to the names in VAR, except that they end with "1"--thus, TEOZZ in VAR becomes TEOZZ1 in NEWVAR.

4) CONTRL:

Common CONTRL includes those important control variables needed by most of the subroutines. It is used in all the physical subroutines, and most of the other routines as well. Its variables are, however, modified only in MAIN. Variables:

ZTZZZ	The current time in the run (in seconds)
DT	The current timestep, Δt (in seconds)
NT	The number of the current timestep
IT	The current number of iterations taken at this timestep
NIT	Total number of iterations taken in the calculation, so far

IBATCH	Index which determines whether we are in batch or interactive mode:
	IBATCH = Ø ==> interactive mode
	IBATCH = 1 ==> batch (card-reading) mode

INEWT	Index for type of calculation which physical subroutines should do: INEWT = Ø ==> make normal calculation INEWT = 1 ==>make new timestep initializations. Check for oxygen starvation; compute the new temperature profile for TMPW and TMPO; take the just-calculated value to be the value to which we add the appropriate new increment (to be found), to find the new integral value. INEWT = 2 ==> For the zeroth iteration at the new timestep, skip calculations--just check the adequacy of the values found by EXTRAP--are they within reasonable physical bounds?

METHOD	Index for numerical method currently in use
	METHOD = Ø ==> Gauss-Seidel (G.S.)
	METHOD = 1 ==> Newton Super Fast (NWSF)
	METHOD = 2 ==> Newton (NWTN)

IVRSN(i)	Index which selects the version of the i th physical subroutine which is user-selected at run-time. $i \leq 20$, but at present is limited to <u>two</u>: If IVRSN(1) = 1, TMPOØ1 will be used " (1) = 2, TMPOØ2 " " " " (2) = 1, ABSRB1 " " " " (2) = 2, ABSBV2 " " " " (2) = 3, ABSRB3 " " "
ZTSTOP	Time at which run terminates.
DTINIT	Size of the initial timestep and maximum size of all following timesteps.
IOUTP	A flag telling whether the timestep has been cut down for output. 1 ==> DT was cut for output 0 ==> DT was not cut for output
IIGNT	A flag giving the status of the most recent ignition. 1 ==> An ignition has just occurred and we have interpolated the time back to the time of ignition 0 ==> An ignition occurred in the previous timestep and we have now reconverged at the timestep we had originally reached -1 ==> No ignitions have occurred in the recent past.
TCONV	The time at which to reconverge assuming an ignition has occurred.
DTOLD	Stores the value of DT should it be temporarily modified for output or ignition.

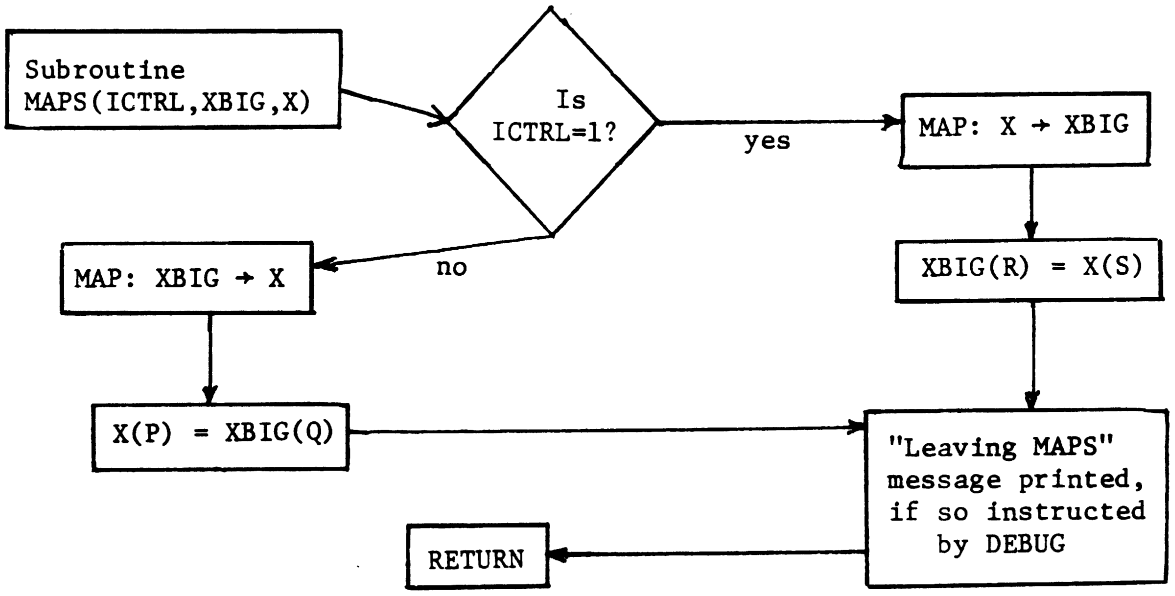
4) IO:

Common IO includes those parameters which control input and output in the program. This common must be included in any subroutine which inputs or outputs information. Common IO is most often used in the control, interface, and input/output routines, but is also used in certain numerical and physical subroutines. Its parameters are initialized in the data block.

Parameters:

IRTTY	Number of the device on the computer used when reading

	information to the terminal (currently 5)
IWTTY	Number of the device on the computer used when writing (outputing) information to the terminal(currently 6)
IRDSK	Number of the device on the computer used when reading (from a disk file) the files linked during this run of the program (currently 21)
IWDSK	Number of the device on the computer used when writing (outputing) information to a disk file (currently 19)
WRTTY	Data from the run of the program is output to the terminal every WRTTY seconds
WRDSK	Data from the run is output to the disk file every WRDSK seconds
ISPOUT	Index controlling output format Ø ==> Long form (table) output to the terminal and to the disk file 1 ==> Short form (column) output to the disk file, long form (table) output to the terminal



Subroutine MAPS(ICTRL,XBIG,X)
Is ICTRL=1?
yes
MAP: X → XBIG
XBIG(R) = X(S)
no
MAP: XBIG → X
X(P) = XBIG(Q)
"Leaving MAPS" message printed, if so instructed by DEBUG
RETURN

Fig. 24. Flow diagram for the interface subroutine MAPS (part of the data structure). MAPS is called by MAIN, DEBUG, WRIT, CALS, and/or NWSTAT. XBIG is any of the large (dimension=500) arrays, such as VAR, OLDVAR, etc. X is any of the small (D=100) arrays, such as X, XØ, etc. P = number of each variable in JCOR. Q = its location in /VAR/. R = location of each variable in /VAR/ that is in JCOR. S = its location in JCOR.

IFNOUT	Index to suppress convergence data printed to the terminal when in Newton's method mode (suppresses the FNORMS). ∅ ==> Print the information 1 ==> Suppress the information
LINES	Gives the count of the number of lines in the short form output in order to insure $\leq$ 55 lines per page of output for proper page skipping.

5) MAP:

Common MAP includes those arrays used in the mappings between the maximum possible set of physical variables, as given in common VAR (500 as of now), the set which actually exists in any one run (54 in a "standard" run, for one room with one wall, one vent, and two objects), and the set of variables in the system, to be checked for convergence (usually 40 or less, in the standard).

This common is used in the control routines, in the interface subroutines, in the numerical subroutines, and in certain Input/Output routines. It never appears in the physical subroutines.

Arrays:

NVAR	The number of physical variables existing in a given run (the length of JCOR)
IPTR(2,2)	Index listing the positions of the bottom and top of JCOR and ICOR inside the array JBLK (see below) (1,1) ==> Position of bottom of JCOR (1,2) ==> Position of top of JCOR (2,1) ==> Position of bottom of ICOR (2,2) ==> Position of top of ICOR
JBLK(600)	Includes arrays JCOR and ICOR, with ICOR located above JCOR in JBLK. JCOR points to the set of variables in VAR which are actually defined for any one run. For example, while we allow for up to 20 room variables, and up to 5 rooms, we have only 15 room variables and only 1 room in the standard run. Thus, the 100 possible variables collapse into 15. The first 15 spots of JBLK point to those 15 variables. ICOR lies above JCOR, and points down to some of the

variables in JCOR. Certain of the physical variables are negligibly small; others are constant; still others are dependent, and are used for no further calculations. We wish to exclude all these from the numerical packages. ICOR points to those variables in JCOR which have <u>not</u> been thus excluded. It is updated quite often during a run.

VMIN(500)	VMIN is structured parallel to common VAR; it defines the minimum significant change for each of the physical variables. It is used in the creation of ICOR, as described above. The values of VMIN are given at the end of DATA BLOCK.

6) DBUG:

Common DBUG includes those flags which control the options of the debugging subroutines, and such auxiliary arrays as are necessary for the routine. When activated, the debugger gives the user added flexibility while running the program, so as to track down problems which may arise.

This common is used in the control, numerical and interface subroutines. It is also used in INIT and in one physical subroutine.

Flags and parameters:

INSYS(100)	An array "parallel" to JCOR. Its elements equal 1 if an element of JCOR is in the system (i.e., is in ICOR), and equal Ø if it is not. This array is useful when listing the variables in the system, and the variables entering and leaving the system.
INSYSP(100)	Similar to INSYS, except that it gives those elements in the system at the previous calculation of ICOR
ITER	Index which determines whether every iteration should be output: if debugger is activated (NDBG = 1) Ø ==> No output each iteration 1 ==> Output each iteration
TBGASK	Time at which to reask the debug questions, if that option is chosen
LISTS	Index for output of a list of the variables in ICOR (in the system), if debugger is activated (NDBG = 1) Ø ==> No list output 1 ==> Output list, each time it is recomputed

LISTEL	Index for output of a list of the variables entering and leaving the system, if the debugger is activated (NDBG=1) Ø ==> No lists output 1 ==> Output lists, each time they are recomputed
NDBG	Main control index for debugger Ø ==> Debugger not now activated 1 ==> Debugger is activated. Do all options requested by the user.
IHELP	Index to inform control routines that user is in trouble Ø ==> No trouble 1 ==> Trouble. Trip debugger.
LEAVE	Index to print out message after leaving a subroutine, if debugger is activated (NDBG = 1) Ø ==> No "leaving" messages 1 ==> "Leaving" messages printed
BUGT	Time at which debugger is to be activated
TTTY	Output interval to terminal (in seconds) to be in effect after the debugger is activated
TDSK	Output interval to disk file (in seconds) to be in effect after the debugger is activated.

7) NUMERC:

Common NUMERC includes those arrays and matrices needed in the Newton-Raphson numerical routines. It is used exclusively for the numerical subroutines.

Arrays:

XJCB(60,60)	Contains the Jacobian matrix
UL(60,60)	Contains the decomposed Jacobian matrix
IPS(100)	Contains the pivoting matrix
SCALE(100)	Contains the scale factors which normalize the variable array in ICOR

8) POINTR:

Common POINTR contains the set of pointers and parameters which describes the setup of the building chosen by the user. The pointers and indices are set up in the input routines, and used in the output, interface, and physical subroutines. Default values for these indices are given in the DATA BLOCK.

Note that the order of the elements MR, MO, MV, MW, and of the elements NR, NO, NV, NW may not be changed lightly. The present order is required by subroutine SETJ.

Indices:

MR, MO, MV, MW	Maximum values that can be input--see VAR, above
NR, NO, NV, NW	Number of rooms, objects, vents, and walls input by the user for a given run
KRO(KO)	Correspondence pointer giving the room number KR in which object KO is located
KNO(KR)	Correspondence pointer giving the number of objects found in room KR
KNOS(KR)	Correspondence pointer giving the total number of objects found in rooms 1 to (KR-1)
KRV(KV,JS)	Correspondence pointer giving the room number KR in which vent KV, side JS, is located
KRW(KW,JS)	Correspondence pointer giving the room number KR in which wall KW, side JS, is located

9) CONST:

Common CONST includes those physical quantities which are indeed constant (i.e. the same regardless of the room, object, vent, and wall in which they are used). As such, its components are unindexed, and include mostly constants of nature.

This common is used in the physical and input subroutines, exclusively. It is initialized in the DATA BLOCK.

Constants: G, SIGMA, PI, CP

10) ROOM:

Common ROOM includes those geometric and physical parameters indexed by room. The parameters in this common are input through the input subroutines and used exclusively in the physical subroutines. Default values for these parameters are given in the DATA BLOCK.

Variables:

ZLRZX(KR), ZLRZY(KR), ZHRZZ(KR), ZKAZZ(KR), VMAZZ(KR), as defined in the Dictionary, Appendix B.

10) OBJECT:

This common includes those geometric and physical parameters which are indexed by object. The state flag ISTAT is used in the control, input, output, interface, and physical routines. The other parameters in this common are input through the input subroutines and used exclusively in the physical subroutines. Default values for these parameters are given in the DATA BLOCK.

Parameters:

ISTAT(KO)	Index giving the current burn status of the object KO. These are listed in Table II, section 4, above. For ISTAT=5, there are subcategories, given by KFTYP.

ZXOZZ(KO)	EB(KO)	ALPHA	FCO(KO)
ZYOZZ(KO)	BETA(KO)	ZTIG(KO)	FS(KO)
ZHOZZ(KO)	CHI(KO)	ZTPYR(KO)	FH2O(KO)
ZNOZZ(KO)	ZMOZO(KO)	ZKOIG(KO)	T
ZJOZZ(KO)	ZRFZO(KO)	ZKOPY(KO)	ZLAMDA(KO)
ZGOZZ(KO)	ZRFZM(KO)	XGAMMA(KO)	TMFGZ(KO)
ZCOZZ(KO)	QF(KO)	XGAMAS(KO)	ZUFZZ(KO)
VMOZZ(KO)	QVAP(KO)	ZRFZI(KO)	ANGV(KO)
	ZOAZZ	FCO2(KO)	ANGH(KO)

All of these are described in the Appendix B dictionary. We also have:

KFTYP(KO)	Index describing type of flaming fire (when ISTAT=5): 1 ==> growing fire 2 ==> pool or ignited fire 3 ==> burner fire

AO(KO)
BO(KO)
NOB(KO)
	Intermediate variables internal to TMPO∅2, which need to be calculated just once in a run; by going into COMMON, they need not be uselessly recalculated each time.

ZKO(20,KO)	Temperature profile of (heated) object KO, from the surface in.
ZKO∅(20,KO)	Same as ZKO, but for previous timestep
AFIRE(KO)=A	The fire-spread rate parameter A in GFIR∅3 (in m/sec)

11) CVENT:

Common CVENT includes those geometric and physical parameters indexed by vent. The parameters in this common are input through the input subroutines and used exclusively in the physical subroutines. Their default values are given in the DATA BLOCK.

Parameters: ZBVZZ(KV), ZHVZZ(KV), ZHTZZ(KV), CD.

12) WALL:

This common includes those geometric and physical parameters indexed by wall. The parameters in this common are input through the input subroutines and used exclusively in the physical subroutines. Their default values are a given in the DATA BLOCK.

Parameters:

ZNWZZ(KW), ZJWZZ(KW), ZGWZZ(KW), ZCWZZ(KW), VMWZZ(KW), ZOWZM, ZOWZN, AW(KW), BW(KW). Parameters used in the computations in TMPWØ1.

N(KW)	The number of points at which the temperature of wall KW is calculated, in TMPWØ1. (Thus, the wall is divided into N-1 slabs.)
ZKW(I,KW)	Temperature of wall KW at the inner boundary of the I^{th} slab. Thus ZKW(1,KW) is the temperature of the inner surface.
ZKWØ(I,KW)	Same as ZKW(I,KW) but used to store the values obtained at the previous timestep.

APPENDIX 6.1

The subroutines of CFC V can be conveniently grouped by their functions. The following is a list of the subroutines of CFC V, grouped by function. This ordering corresponds to that used when describing the location of the commons, given above. It is slightly different from the grouping shown in fig. 8, but the differences are semantic only.

Control Routines:	MAIN, DELTAT, RESETI, NWSTAT, TIGN
Input Routines:	DATA BLOCK, INIT, INPUT3, ALTINP, DISP, COPINP, VERIFY, RECAP
Output Routines:	WRIT, WRITØ3, LOOKUP, DEBUG, LIST
Numerical Routines:	EXTRAP, NUMER, JACB, NWTN, MSLV, DECOMP, SOLVE, SING, CONV
Interface Routines:	SETJ, SETI, MAPS, CALS, SELSUB

Physical Routines	RDNO, RNWO, RNLO, RNPO, RNLH, RNLV, RNFF, RDNP, RDNW, RDNL, CNVL, CNVW, HFIRØ1, GFIRØ3, PFIR, BFIRØ1, LAYR, ABSRB1, ABSRB2, ABSRB3, PLUM, PLHT, TMPW, TMPO, TMPF, VENT, FLOW

APPENDIX 6.2

In Gauss-Seidel, we use CALS, with input variables XK and output variables XK1. In order to run the physical subroutines, we have to have the values in common /VAR/. Hence the first step is to "unpack" the relatively densely packed array XK into the sparse array XIN(500); the latter array is just the array in /VAR/, only the name being different. (In file VAR, common /VAR/ is the set of arrays TELZR(5),..., comprising one "super array" of 500 items. In file VAR1, exactly the same array is called XIN.)

The physics subroutines produce variables ending in "1"--eg. TELZR1. However, the calls to these subroutines (from CALS) are of the form CALL(...,TELZR(KR)), so that TELZR is returned without the suffix 1, and placed directly into /VAR/. The final step is to repack (again via MAPS) the new array XIN into XK1. XK and XK1 are then compared so as to check on convergence. If converged, XK1 is placed in X. If not converged (and provided other criteria are satisfied), XK is thrown away, XK1 placed in XK, and we start over.

In NEWTON, we use CALS1. This produces calls to the subroutines of the form CALL(...,TELZR1(KR))--i.e., the suffix 1 is retained. These are placed in common /NEWVAR/ (in file NEWVAR). The array NEWVAR is also called XNEW (in file NEWVAR1). Thus XK → XNEW → Physics → XNEW → XK1.

Finally, previous timestep values, such as TELZRP, are placed in common /OLDVAR/. The alternative name for these variables is of course XOLD (in file OLDVAR1).

7) Input and Output.

There are two options for the input of data: batch and interactive. The latter is easier to use, partly because the questions are shown sequentially on the screen (by subroutine INPUTØ3). The user is presented with default data (in blocks), and can change as much or as little of it as he desires. This input is then reviewed (via subroutine DISP), and an opportunity to correct any errors or change your mind is given, before going on to the next block of data. After all the data is input, a final opportunity is given to change the information.

When the user is satisfied that he has input all the correct information for a run, all that information is displayed one final time (by RECAP). During input, if a syntax error is made, this will generally be caught by the program, and the question reasked. Thus, the input is relatively flexible and forgiving. It is necessarily less so for batch processing, of course; in punching the cards, one must carefully anticipate each question, and have the correct answer ready. Batch mode is taken care of in subroutine ALTINP. Rather than suppress the questions to a nonexistent reader, they are merely sent to a "garbage" file. There's also no provision for error messages, nor for reaction to syntactical errors in input. Finally, the "YOU ARE IN TROUBLE" message which appears when convergence is not forthcoming, does not appear--instead, the calculation simply terminates.

One of the options available in the interactive mode, is to invoke subroutine DEBUG (at any preset time); it is of course principally useful to those users who are actually working on the program itself. In case of difficulty during execution of the program, DEBUG will be automatically invoked, permitting the (interactive) user to stop or to proceed in a number of ways. This cannot be done in batch mode, of course.

Output is in two modes: first, results of the calculations are displayed on the screen, periodically (the default period is 20 seconds, but this can be changed by the user at run time). This allows run-time examination of how the run is proceeding. Second, the results are also stored on a disk file, which can then be printed. The interval for storage of results on disk is independent of that for display; the default value is 10 seconds, but this, too, can be altered to suit the user's needs at run time.

The user has a choice of two formats for the printed output: first, the "long form", which is identical to that shown on the screen, and which gives the values of most of the variables involved. Alternatively, a short form is available, which gives the values of just <u>eight</u> selected variables. These appear in a single row, and thus the output is much more compact. The eight variables displayed can be chosen at run-time by the user; if no choice is made, the eight <u>default</u> variables displayed are: the layer temperature T_L; the surface temperature of object 2, $T_s(2)$; the ceiling temperature T_W; the layer depth h_L; oxygen (mass) concentration $Y(O_2)$; mass efflux from room, $\dot{m}_u$; rate of pyrolysis of original fuel item, $\dot{m}_f(1)$; and radius of fire of object #1, $R_f(1)$.

These two output formats are shown in Appendix C. Some more discussion may be found in Mitler (28), although that applies, strictly, only to the Mark IV version.

We also output comments, when various interesting things happen. Thus, the ignition of an object is announced together with the time this occurs. Similarly, when oxygen starvation begins and ends for a burning object, and the burnout of an object.

Finally, we might mention here that the present compiled program length is about 120 kilobytes (30,000 words). The program is at present divided into 13 files: CARK5, DARK5, FARK5, IARK5, LARK5, MARK5, NARK5, PARK5, RARK5, SARK5, TARK5, VARK5, and WARK5. These are abbreviated as C, D, F, I, L, M, N, P, R, S, T, V, and W, respectively. They contain the subroutines corresponding to:

C =	convective heat transfer
D =	discovery--i.e. state changes of objects, such as ignitions, etc.
F =	fire
I =	initializations, input
L =	layer
M =	Main control program
N =	numerics
P =	plume
R =	radiation
S =	structure and interface routines, such as CALS, SETJ, MAPS, etc.
T =	temperature of walls, objects, etc
V =	vents
W =	output (WRIT, etc.)

V. SUBROUTINES FOR THE PHYSICS.

In this section, all the subroutines giving the physics used in Mark V are written up. For each subroutine the file in which it appears and the calling subroutine is given*. Then there is a brief explanation of the assumptions and approximations made, and (generally) an indication of where the equations are derived. Then the input and output variables are listed; for the former, the subroutine where they are found or calculated is given, and any explanatory remarks. Finally, the equations on which the subroutine is based are given. In the variable lists, no definitions will be given, since these are already given in the Dictionary (Appendix B.1).

GFIRØ3

Eqs. by H. Mitler, Sept. 1980

This subroutine is in file F; it is called by CALS (or CALS1).

Description. This subroutine describes the burning of a horizontal fuel slab, ignited at the center of its top surface. The fire grows slowly until the entire surface (including the sides) is burning. The pyrolysis rate, the resulting burning and energy-release rates, the fire radius and the flame shape are all calculated, as a function of time. (It is similar to GFIRØ1 in CFC IV and to FIREØ2 in CFC III.)

The pyrolysis rate is taken to be proportional to the net heat flux to the surface and to the (instantaneous) burning area. The net heat flux is the radiation incident from all sources (principally the flame) plus the convective heating from the flame, minus the reradiation. For small flames convective heating is substantial, but rapidly falls in importance as the flame grows. Theoretical expressions for it exist, but they were felt to be too complex to include. Instead, the reradiation is simply cut down for small flames in such a way that the observed growth rate of pyrolysis is followed. This _ad hoc_ procedure turns out to be very simple. The burning surface is taken to be at $700^{\circ}K$ (polyurethane foam is our default material). Note that if we reradiated at $700^{\circ}K$ without adding a convective contribution, the flame

* This will generally be CALS. When that is written, it will be understood that it will be CALS1, when in NWTN mode.

would self-extinguish when small. We ignore the energy lost by diffusion into the slab. The pyrolysis rate is also modified (in an arbitrary way) to take into account the smooth cessation of pyrolysis as the fuel nears burnout.

The flame is modeled by a right circular cone of hot (grey) gas, of semiapex angle ψ. The initial value of ψ is $\psi_o=30^o$. The cone is homogeneous, and has the temperature $T_f=1260^oK$, the absorption coefficient $\kappa=1.55m^{-1}$ (values obtained by L. Orloff and G. Markstein at FMRC).

We do not yet know how the flame is affected by the vitiated layer ; as in previous versions, we assume that the flame continues to burn as fully as in the open air, so long as the amount of air entrained into the lower part of the plume (below the interface with the hot layer) provides enough oxygen. That presumes that all of the air entrained therein is incorporated into the flame, and that none of the oxygen-vitiated gases from the upper part of the plume (that traveling through the hot layer) contribute any oxygen. These seem like a fairly reasonable set of assumptions, and the obvious errors are (partly) compensating ones.

When the layer is so low that the amount of air entrained is insufficient for "full" burning, we call this "oxygen starvation", and reduce the burning rate appropriately. We assume constant volumetric heating, and therefore must reduce the cone volume. This is done by increasing ψ. ψ is also increased near burnout, when the burning rate is reduced.

We have--erroneously no doubt--assumed that when the layer actually covers the burning item, it continues to pyrolyze as before, even though there is no flame. And that if the layer were to rise, the flame would reappear. We therefore assume that the (immersed) surface stays at its previous temperature and pyrolyzes due to the radiative flux it receives from the (accessible part of the) layer, and from the extended ceiling (this is the flux ϕ_+). We ignore convective heating or cooling by the layer, and reradiation from the surface. Even when the surface is not quite covered, but ϕ_{net} (see calculations below) would fall to a low or negative value, we take ϕ_+ to be the minimum the surface receives. When it is nearly burned out, however, it must receive still less-- we then assume an effective flux ϕ_{eff} such that it will burn up with an e-folding time of about two seconds: $\dot{m}_f \doteq -m_f/2$.

The fire radius grows until it reaches a maximum such that the resulting area is the area of the top plus sides of the slab. This equivalent maximum radius R_m is of course larger than the radius corresponding to the top surface above. The spread rate until $R=0.95R_m$ is given by an expression which yields approximately exponential growth (as observed) but avoids the divergence inherent in older versions of this routine. Beyond that point, the spread rate is taken to slow down, as shown below. See pp. 60-68 of Technical Report #34 (Reference 3) for detailed discussion. The default data used here is for flexible polyurethane foam #7004.

<u>Output</u>

$m_f \equiv$ ZMOZZ	$\dot{E}_f \equiv$ TEOZZ
$\dot{m}_f \equiv$ TMOZZ	$R \equiv$ ZRFZZ
$\psi \equiv$ PSI(KO)	Semiapex angle of cone modeling the flame (Actually the tangent of ψ is calculated and carried in COMMON as TPSI)

Input Required from Common	Calculated in Subroutine		Remarks
$\Sigma_p \dot{q}''_{pf} \equiv$ FQPOR(KO)	RNPO	RDNO	Here, object KO is the burning one; the impinging fluxes are from <u>all</u> flames (plumes), and all walls.
$\dot{q}''_{ef} \equiv$ FQLOR(KO)	RNLO		
$\Sigma_w \dot{q}''_{wf} \equiv$ FQWOR(KO)	RNWO		
$\dot{m}_p \equiv$ TMPZZ(KO)	PLUM		
$m_f \equiv$ ZMOZZ	GFIR		Remaining fuel mass from previous timestep
$R_m \equiv$ ZRFZM(KO)	INPUT		
$R \equiv$ ZRFZZ(KO)	GFIR		Radius of fire from previous timestep
$R_o \equiv$ ZRFZO(KO)	INPUT		
A(KO) $\equiv$ AFIRE	DATA		Spread-rate parameter; 0.0109 m/sec for PU foam
$\lambda \equiv$ XGAMMA	INPUT		Default value is for PU foam, 14.45
$\chi \equiv$ CHI(KO)	INPUT		Default value is 0.65 (for P.U. foam)
$H_v \equiv$ QVAP(KO)	INPUT		2.054×10^6 joules/kg, for PU foam

Input Required from Common	Calculated in Subroutine	Remarks
$H_c \equiv$ QF(KO)	INPUT	2.89×10^7 joules/kg for PU foam
$m_o \equiv$ ZMOZO(KO)	INPUT	When a material is a composite which is partly inert, it is expedient to ignore that part, and only list the mass of the pyrolyzable part; we would thus take m_o = pyrolyzable fraction of m_{total}.
$T_f \equiv$ ZKFZZ $\equiv$ T	INPUT	1260°K
$T_p \equiv$ ZKOPY(KO)	INPUT	600°K for PU foam
$T_a \equiv$ ZKAZZ	INPUT	Default value, 300°K
$c_p \equiv$ ZCFZZ(KO)	INPUT	
$\kappa \equiv$ ZUFZZ(KO)	INPUT	1.55m^{-1}

Calculations

$\dot{q}''_f = \dot{q}''_{Lf} + \sum_w \dot{q}''_{wf} + \sum_p \dot{q}''_{Pf}$	Radiative flux impinging on fire KO from all hot sources
$\dot{q}''_{rr} = \sigma T_s'^4 = \min[13200, 21700\ \kappa R]$	Effective reradiated flux (this mimics including the effect of initial convective heating of the surface by the flame).
$\phi_{net} = \dot{q}''_f - \dot{q}''_{rr}$	Net flux incident on fire KO, when flame height is not significantly reduced.
$\phi_+ = \dot{q}''_{LF} + \sum_w \dot{q}''_{wf}$	Minimum flux incident on fuel surface, when not near burnout.
$\phi_{eff} = m_f H_v / 2\pi R^2$	Effective flux seen by (fictitious) entire surface πR^2, when fuel is nearly exhausted.
$\phi_{max} = \max(\phi_{net}, \phi_+)$	
$\phi = \min(\phi_{max}, \phi_{eff})$	Flux incident on surface of fire KO.
$C = \phi / \sigma T_f^4$	

$$\dot{R} = \begin{cases} AC(1 + C/2 + C^2/3) & R \leq 0.95R_m \\ (R_m - R)/10 & R > 0.95R_m \end{cases}$$

Rate of change of fire radius (i.e. spread rate)

Calculations (cont'd)	Remarks
$R = R_o + \int_o^t \dot{R}(t')dt'$	Radius of fire
$\dot{m}_\beta = \begin{cases} -\pi R^2\phi/H_v, & \phi>0 \\ 0 & \phi\leq 0 \end{cases}$	Unadjusted rate of change of fuel mass (negative of pyrolysis rate)
$G = m_f/2\dot{m}_\beta$	Convenient definition [for $\dot{m}_\beta<0$]
$\dot{m}_f = \begin{cases} \dot{m}_\beta(1-e^G), & \phi>0 \\ 0 & \phi\leq 0 \end{cases}$	Negative of pyrolysis rate, adjusted for smooth burnout
$m_f = m_o + \int_o^t \dot{m}_f(t')dt'$	Mass of fuel remaining at time t
$\dot{m}_b = \min[-\chi\dot{m}_f, (\dot{m}_p + \dot{m}_f)/\gamma]$	Burning rate, limited by combustion efficiency χ, or by oxygen starvation
$H_c' = H_c - c_p(T_p - T_a)$	Effective heat of combustion
$\dot{E}_f = -\dot{m}_b H_c'$	Negative of power output by combustion
when $\dot{m}_b < -\chi\dot{m}_f$,	Semiapex angle of fire cone, when burning is oxygen-limited
$\tan\psi = \chi\tan\psi_o\|\dot{m}_f\|/\dot{m}_b$	Note that $\chi\gamma\tan 30^\circ = (0.65)(14.45)/\sqrt{3} = 5.422762$

PFIRØ2

Eqs. by H. Mitler, July 1980

In file F; called by CALS. The name is a contraction of "Pool Fire".

Discussion. This subroutine calculates the rate of pyrolysis of a pool fire -- i.e., a fire of fixed radius. It differs from FIREØ3 (in CFC IV) in that (a) the fuel can have a dinite mass, (b) it can be for any fuel, and (c) the data is read in via INPUT, rather than being given by data statements, in the subroutine itself. The pyrolysis rate is taken to be proportional to the net incoming radiative flux. PFIRØ2 also makes the calculations for a fire ignited during the run, by autoignition: When the entire (horizontal) surface of a combustible material is heated to its ignition temperature T_{ig}, ignition will start at a "hot spot" and spread very rapidly over the surface. This rapid spread was inadvertently done incorrectly in IFIRØ1 (in CFC IV); it was corrected in IFIRØ2, where the radius was taken to increase asymptotically to its maximum value with an e-folding time of 2 seconds. Note, however, that in order to avoid numerical instabilities with pool fires, the same time-dependence for R(t) was used in PFIR. Hence IFIR and PFIR became essentially identical, and they have been combined here; IFIRØ2 therefore does not appear in this report. This subroutine gives the pyrolysis rate, burning rate, energy-release rate, and the effective radius of the burning surface after ignition (which, for a pool fire, occurs at $t_{ig} = 0$). For oxygen-starved fires it also gives the (increased) cone angle; that is one improvement over IFIRØ2. Another is the way in which the slow decay near fuel exhaustion is calculated: it is now so calculated that the decay time is relatively constant, independent of the original fuel mass, room size, etc. A third, and most important one, is in taking the re-radiation into account, as in GFIRØ3.

Output: m_f ≡ ZMOZZ, $\dot{m}_f$ ≡ TMOZZ, $\dot{E}_f$ ≡ TEOZZ, R ≡ ZRFZZ, $\tan\psi$ ≡ TPSI.

Input Required from Common	Calculated or Found in Subroutine	Input Required from Common	Calculated or Found in Subroutine
$\sum_p \dot{q}''_{pf}$ ≡ FQPOR(KO)	RNPO } RDNO	H_v ≡ QVAP(KO)	INPUT
$\dot{q}''_{Lf}$ ≡ FQLOR(KO)	RNLO } RDNO	H_c ≡ QF(KO)	INPUT
$\dot{q}''_{wf}$ ≡ FQWOR(KO)	RNWO } RDNO	m_o ≡ ZMOZO(KO)	INPUT
$\dot{m}_p$ ≡ TMPZZ(KO)	PLUM	T_p ≡ ZKOPY(KO)	INPUT
m_f ≡ ZMOZZ(KO)	GFIR	T_a ≡ ZKAZZ	INPUT
R_m ≡ ZROZM(KO)	INPUT	c_p ≡ ZCFZZ(KO)	INPUT
γ ≡ XGAMMA	INPUT	κ ≡ ZUFZZ(KO)	INPUT
χ ≡ CHI(KO)	INPUT	t_{ig} ≡ ZTIG(KO)	TIGN

<u>Calculations (cont'd)</u>:	<u>Remarks</u>:
$t_s = \frac{100\dot{m}_g H_c \chi (\Delta t)^2}{\rho_a c_p T_a V}$	Time (after ignition) at which fuel flow is on fully (in seconds)
$t < t_s \Longrightarrow \dot{m}_f = -\dot{m}_g[1 - (1-t/t_s)^2]$	Fuel flow rate as a function of time
$\dot{q}''_f = \dot{q}''_{Lf} + \dot{q}''_{wf} + \sum_p \dot{q}''_{pf}$	Radiative flux impinging on fire KO from all hot sources
$\dot{q}''_{rr} = \sigma T'^4_s = 13200[1 - \exp(-1.64\kappa R)]$	"Effective" reradiated flux (see discussion in text of GFIR)
$\phi = \dot{q}''_f - \dot{q}''_{rr}$	Net flux incident on fire KO
If $\dot{q}''_f - \dot{q}''_{rr} \le 0$, however, let	
$\phi = \dot{q}''_{Lf} + \dot{q}''_{wf}$	
$R = R_m[1 - e^{-(t-t_{ig})/2}]$	Radius of fire
$A = \pi R^2$	Area of fire
$\dot{m}_\beta = \begin{cases} -\pi R^2\ \phi/H_v, & \phi > 0 \\ 0, & \phi \le 0 \end{cases}$	Unadjusted rate of change of fuel mass (negative of pyrolysis rate)
$F = \begin{cases} -m_f/2\dot{m}_f, & \phi > 0 \\ 0, & \phi \le 0 \end{cases}$	Convenient definition
$\dot{m}_f = \dot{m}_\beta(1 - e^{-F})$	Negative of pyrolysis rate, adjusted for smooth burnout
$m_f = m_o + \int_o^t \dot{m}_f(t')dt'$	Mass of fuel remaining at time t
$\dot{m}_b = \min[-\chi\dot{m}_f, (\dot{m}_p + \dot{m}_f)/\gamma]$	Burning rate, limited either by combustion efficiency χ or by oxygen starvation
$H_c' = H_c - c_p(T_p - T_a)$	Effective heat of combustion
$\dot{E}_f = -\dot{m}_b H_c'$	Negative of power output by combustion
When $\dot{m}_b < -\chi\dot{m}_f$, $\tan\psi = \chi \tan\psi_o \lvert\dot{m}_f\rvert/\dot{m}_b$	ψ is the semiapex angle of fire cone, when burning is oxygen-limited. Note that $\chi\lambda\tan 30^\circ = \frac{(0.65)(14.45)}{\sqrt{3}} = 5.422762$

BFIRØ1

Equations adapted from PFIRØ1 by John A. Rockett, April 1980.

Coded by John A. Rockett, largely a direct copy of PFIRØ1.

Description: This subroutine calculates the energy output from a gas burner fire. The routine is a specialization of the fixed diameter pool fire routine PFIRØ1. To obviate some numerical problems associated with the rapid initiation of this type of fire, the initial rate of heat release is arbitrarily reduced. The (input) gas flow rate is assumed to build up from zero at zero time to the constant input value at a time TSTART, following a parabola with finite slope at time zero and zero slope at time TSTART. Note that here, of all places, ψ ought to be variable. In this first version, however, we ignore that fact.

Output for the burner, object KO:

$m_f \equiv$ ZMOZZ	Stays at initial value ZMOZZP(KO).
$\dot{m}_f \equiv$ TMOZZ	Never changes from gas burner input flow rate -TMFGZ after t_s.
$\dot{E}_f \equiv$ TEOZZ	
$R_f \equiv$ ZRFZZ	Stays at initial value ZRFZM(KO).

Required Input	From	Remarks
ISTAT(KO)	INPUT	Status index, = 6 if burner is ignited
INEWT	CALS	Control index, = 1 if CALS is initializing
$\dot{m}_g \equiv$ TMFGZ(KO)	INPUT	Final (maximum) gas flow rate
$\gamma \equiv$ XGAMMA(KO)	INPUT	Air/fuel mass ratio for the fuel from that burner

$\dot{m}_p \equiv$ TMPZZ(KO)	PLUM	$\rho_a \equiv$ VMAZZ	INPUT
$\chi \equiv$ CHI(KO)	INPUT	$c_p \equiv$ CP	INPUT
$t \equiv$ ZTZZZ	MAIN	$h_R \equiv$ ZHRZZ	INPUT
$\Delta t \equiv$ DT	MAIN	$W = L_y \equiv$ ZLRZY	INPUT
ZKAZZ	INPUT	$L = L_x \equiv$ ZLRZX	INPUT
$H_c \equiv$ QF	INPUT		

Calculations:	Remarks:
$V = WLh_R$	Volume of room

<u>Calculations</u> (concluded)	<u>Remarks</u>
$t > t_s \Longrightarrow \dot{m}_f = -\dot{m}_g$	Fuel flow rate as a function of time
$\dot{E}_f = H_c \min[-\chi\dot{m}_f, (\dot{m}_p + \dot{m}_f)/\gamma]$	

RDNO

In file R; called by CALS.

<u>Description</u>: Controlling subroutine for radiation to any object (whether burning or not) in room KR. It calls sub-subroutines RNLO, RNWO, and RNPO, which do the actual calculating. It also sets to zero any flux which is less than 0.1 w/m^2, as too small to be physically meaningful. This subroutine is essentially identical to the version in CFC IV. The original version was RADOØ1, found on p. 49 of CFC III.

<u>Output</u> $\dot{q}''_{LO}$ ≡ FQLOR, $\dot{q}''_{WO}$ ≡ FQWOR, $\dot{q}''_{PO}$ ≡ FQPOR from subroutines RNLO, RNWO, and RNPO, respectively.

RNLO

In file R; called by RDNO.

<u>Description</u>: Controlling subroutine for radiation from the layer to a horizontal or vertical surface. It calls RNLH and RNLV, respectively, to get these fluxes.

<u>Output</u>	<u>From Subroutine</u>	
$\dot{q}''_{Lo}$ ≡ FQLOR	RNLH	When object is horizontal
	RNLV	When object is vertical

<u>Input</u>

θ_H ≡ ANGH(KO)	INPUT	Angle (plane) surface of object KO makes with the horizontal (in degrees)

<u>Calculation</u>

If $\theta_H < 45^\circ$,	assume surface is horizontal -- i.e. $\theta_H = 0$, and call RNLH
If $\theta_H \geq 45^\circ$,	assume surface is vertical -- i.e., $\theta_H = 90^\circ$, and call RNLV

RNLH

Equations by H.E. Mitler, April 1977

Modified July 1978, October 1980. In file R; called by RNLO.

<u>Description</u>. Each object and surface in the room radiates to every other object. The two most important radiators are the flame and the hot ceiling

layer, consisting of hot gas and soot. RNLH calculates (approximately) the radiation from the layer to a horizontal surface (the object KO). The room is divided into four quadrants by the point at the center of the surface, and each resulting quadrant of the layer is approximated by the quadrant of an equivalent cylinder -- i.e., one of equal thickness and volume. Version IV differed from that described on p. 53-4 of CFC III in including the case where the target surface is burning. The present version differs from that in Mark IV only in having a variable ψ and (of course) in using the LIMITS procedure. The derivation is given on page 42 of TR34.

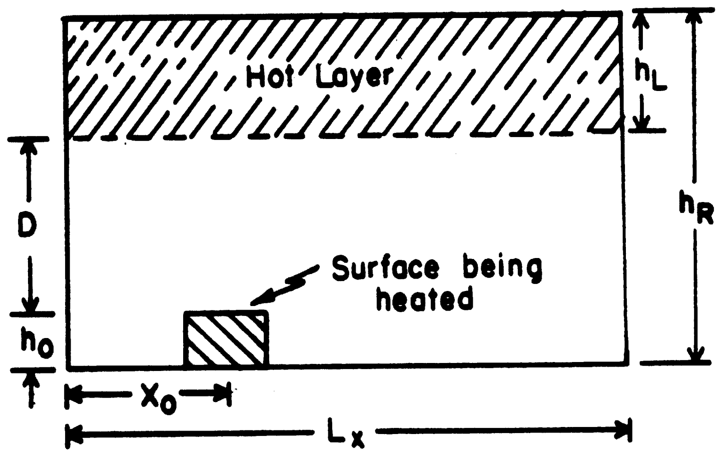


Fig. 25. Side View of Room

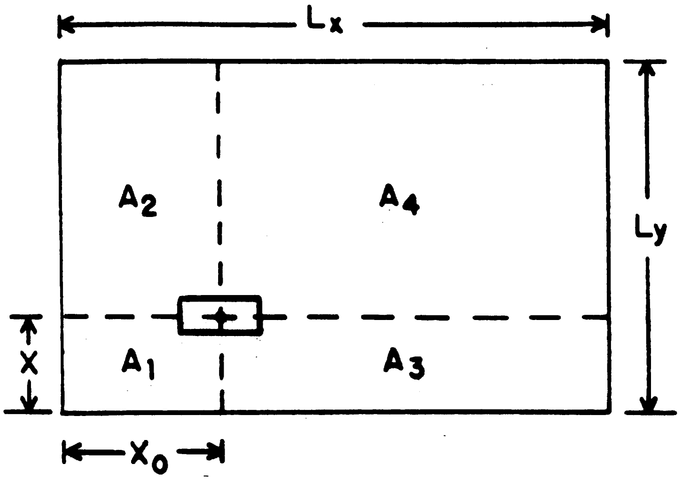


Fig. 26. Top View

Output

$\dot{q}''_{LO} \equiv$ FQLOR(KO)

Input Required from COMMON	Found or Computed in
$x_o, y_o \equiv$ ZXOZZ,ZYOZZ	INPUT
$h_R \equiv$ ZHRZZ	INPUT
$h_o \equiv$ ZHOZZ	INPUT
$h_L \equiv$ ZHLZZ	LAYR
$L = L_x \equiv$ ZLRZX	INPUT
$B = L_y \equiv$ ZLRZY	INPUT

Input Required from COMMON	Found or Computed in
$\sigma \equiv$ SIGMA	DATA BLOCK
$T_L \equiv$ ZKLZZ	LAYR
$k \equiv$ ZUFZZ	DATA
$\kappa \equiv$ ZULZZ(KR)	ABSRB
$R_f \equiv$ ZRFZZ(KO)	FIRE
$\tan \psi \equiv$ TPSI(KO)	FIRE

<u>Calculations</u> <u>Remarks</u>

$$D = h_R - h_o - h_L \qquad \text{Distance between layer and object surface}$$

$$\left.\begin{aligned} A_1 &= x_o y_o \\ A_2 &= x_o(B - y_o) \\ A_3 &= (L - x_o)y_o \\ A_4 &= (L - x_o)(B - y_o) \end{aligned}\right\} \text{Areas into which ceiling is partitioned}$$

$$\left.\begin{aligned} R_i &= \sqrt{\frac{4A_i}{\pi}} \\ Z_{oi} &= (R_i^2 + D^2)^{1/2} \\ Z_{li} &= [R_i^2 + (D + h_L)^2]^{1/2} \\ S_i &= \frac{2Z_{oi}^2}{R_i^2}(h_L + Z_{oi} - Z_{li}) \end{aligned}\right\} \begin{aligned} &\text{Equivalent radii of ceiling quadrants} \\ &\text{Computation parameters} \\ &i = 1, 4 \end{aligned}$$

$$e_i = 1 - e^{-S_i\kappa} \qquad \text{Effective emissivity of ceiling layer}$$

$$\phi_{LO} = \frac{\sigma T_L^4}{4}\sum_{i=1}^{4} e_i\left(1 + \frac{D^2}{R_i^2}\right)^{-1}$$

$$\dot{q}''_{LO} = \phi_{LO}$$

Radiative flux from the hot layer to the point (x_o,y_o) on the surface of object KO, when it is not burning.

When the object <u>is</u> flaming, then

$$\dot{q}''_{LO} = \phi_{LO}\frac{e^{-\tau} + \sin\psi}{1 + \sin\psi}$$

where $\tau = kR_f$

<u>Nomenclature</u>

The letters RN in the code refer to radiation.

The third letter identifies its source:

L - layer	P - plume
F - fire	C - ceiling
W - wall	etc.

The fourth letter specifies the nature of the receiving surface (when that is appropriate):

H - horizontal	O - object (assumed horizontal!)
V - vertical	W - wall
etc.	

RNLV

Equations by Henri E. Mitler, February 1977.

In file R; called by RNLO.

Description. Comments are the same as for RNLH. This subroutine gives the irradiation of a vertical surface, however. The geometry is as shown in Fig. 25 of RNLH and in Fig. 27 below, which is the top view of the room. The exposed surface is at 0, and for the orientation shown, given by the angle γ (i.e., the azimuthal angle ϕ, in spherical coordinates) it "sees" the area bounded by the line segments QR, RC, CB, and BQ -- i.e., bounded by QRCBQ. The area B_1 is bounded by QOPBQ, B_2 by OPCRO. POR is a right angle. Note that γ takes on any value between 0 and 2π. B_1 and B_2 are given by the areas shown in column 2 of Table III below, which refer to the areas displayed in Fig. 28. In the latter figure, $0 < \beta < \pi/2$. The values S_i are given in Table IV.

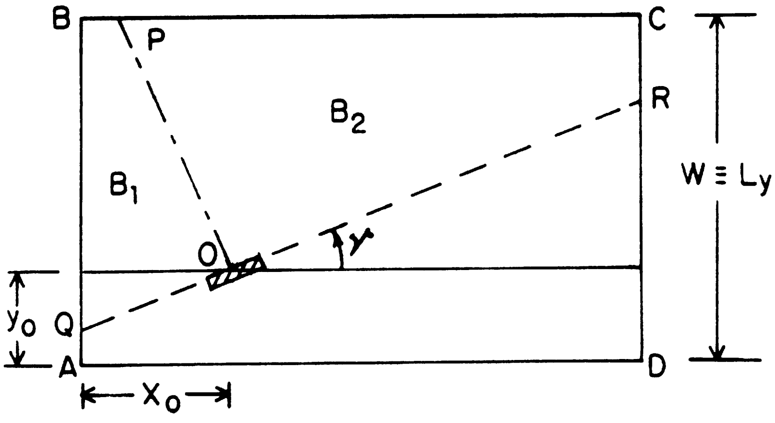


Fig. 27: Top View of Room. Vertical Surface is at the point 0.

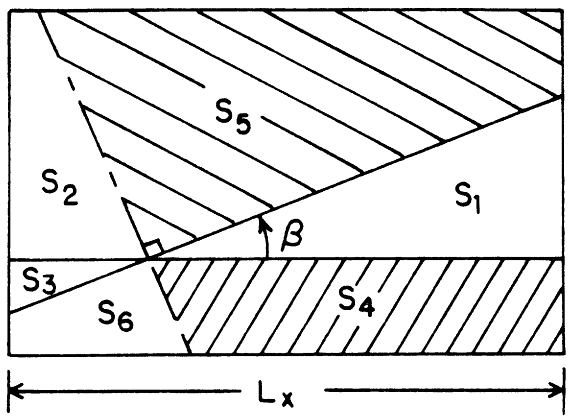


Fig. 28: Definition of the areas S_i.

TABLE III: The areas B_1 and B_2 in terms of S_i; β in terms of γ.

γ	Areas B_2 & B_1, respectively	β
$0 \le \gamma < \frac{\pi}{2}$	S_5, $S_2 + S_3$	γ
$\frac{\pi}{2} \le \gamma < \pi$	$S_2 + S_3$, S_6	$\gamma - \frac{\pi}{2}$
$\pi \le \gamma < \frac{3}{2}\pi$	S_6, $S_1 + S_4$	$\gamma - \pi$
$\frac{3}{2}\pi \le \gamma < 2\pi$	$S_1 + S_4$, S_5	$\gamma - \frac{3}{2}\pi$

TABLE IV: The areas S_i.

$S_1 = \frac{1}{2}(L-a)^2\tan\beta$ if $\tan\beta \leq \frac{W-b}{L-a}$	$= (L-a)(W-b) - \frac{1}{2}(W-b)^2\cot\beta$ if $\tan\beta > \frac{W-b}{L-a}$
$S_2 = a(W-b) - \frac{1}{2}(W-b)^2\tan\beta$ if $\tan\beta \leq \frac{a}{W-b}$	$= \frac{a^2}{2}\cot\beta$ if $\tan\beta > \frac{a}{W-b}$
$S_3 = \frac{a^2}{2}\tan\beta$ if $\tan\beta \leq \frac{b}{a}$	$= ab - \frac{b^2}{2}\cot\beta$ if $\tan\beta > \frac{b}{a}$
$S_4 = b(L-a) - \frac{b^2}{2}\tan\beta$ if $\tan\beta \leq \frac{L-a}{b}$	$= \frac{1}{2}(L-a)^2\cot\beta$ if $\tan\beta > \frac{L-a}{b}$
$S_5 = L(W-b) - S_1 - S_2$	
$S_6 = bL - S_3 - S_4$	

<u>Output</u>	<u>Definition</u>
$\dot{q}''_{LO} \equiv$ FQLOR(KO)	Radiative flux to (vertical) object KO, from the hot layer (in w/m^2).

<u>Input required from COMMON</u>

Exactly the same as in RNLH, except that we do <u>not</u> need k, R_f, or tan ψ. However, we <u>do</u> need

$\gamma \equiv$ ANGV(KO)	INPUT	Orientation angle of exposed surface of KO (see fig. 27) (Not to be confused with XGAMMA!)
(Moreover, $B \equiv W \equiv L_y$)		

<u>Calculations</u>

$a,b = x_o, y_o$

$R_1 = \sqrt{4B_1/\pi}$, $R_2 = \sqrt{4B_2/\pi}$ — Radii of equivalent quadrants, yielding the same radiating volume and area as the actual areas B_1 & B_2.

$D = h_R - h_o - h_L$

If $D > 0$:

$$\alpha_i = \tan^{-1}\frac{R_i}{D}$$

<u>Output (cont'd)</u>

<u>Definition</u>

$$\beta_i = \alpha_i - \frac{1}{2}\sin 2\alpha_i = \tan^{-1}\frac{R_i}{D} - \frac{DR_i}{D^2 + R_i^2}$$ $i = 1,2$

$$Z_{oi} = \sqrt{R_i^2 + D^2}$$

$$Z_{1i} = \sqrt{R_i^2 + (D + h_L)^2}$$

$$S_i' = \frac{2\pi}{\beta_i}\left[(D + h_L)\ln\left(\frac{Z_{1i} + R_i}{D + h_L}\right) - D\ln\left(\frac{Z_{oi} + R_i}{D}\right)\right]$$

Effective beam lengths in the layer.

$$\dot{q}''_{LO} = \frac{\sigma T_L^4}{2\pi}\sum_{i=1}^{2}\beta_i(1 - e^{-S_i'\kappa}) \text{ watts/m}^2$$

Radiative energy flux incident on the vertical surface, due to the hot layer.

If $D \leq 0$,

Case where the object is <u>in</u> the layer.

$$\dot{q}''_{LO} = \frac{\sigma T_L^4}{4}\sum_{i=1}^{2}\left[2 - e^{-\kappa P_{i1}} - e^{-\kappa P_{i2}}\right]$$

where $P_{ij} = 4\, h_j \ln\left[\frac{\sqrt{R_i^2 + h_j^2} + R_i}{h_j}\right]$ $j = 1,2$

and $h_1 = h_L + D,$
$h_2 = -D.$

RNWO$\emptyset$2

Eq . by H. Mitler, Sept. 1980.

Called by RDNO; in file R.

<u>Description</u>. Every object (burning or not) receives radiation from the walls and ceiling of the enclosure. This subroutine calculates that flux for horizontal surfaces. It differs from RNWO$\emptyset$1 (used in CFC III & IV) in the following ways: (1) The room is broken up into four rectangular sections (with the target object at the corner of each piece) and each piece is then approximated as the quadrant of an equivalent cylinder, as in RNLO. (2) The view factor for each piece is then computed geometrically, rather than being inferred from a previous calculation. (3) The attenuation through the hot layer is found more nearly adequately. (4) The radiation from the <u>cold</u> walls is included here as well, rather than being found in HEAT, TMPO, or the other subroutines which calculate the surface temperature of objects. (5) The corrections to be made for the vent surfaces have been ignored here; this is generally only a minor error, however.

Output

$\phi_{WO} \equiv$ FQWOR(KO)

Input required from COMMON	Found or Calculated in Subroutine	Input required from COMMON	Found in Subroutine
x,y	INPUT	$\lambda \equiv$ ZLAMDA	DATA BLOCK
$L \equiv L_x \equiv$ ZLRZX	INPUT	$\tan\psi \equiv$ TPSI	DATA BLOCK or FIRE
$h_R \equiv$ ZHRZZ	INPUT	$R_f \equiv$ ZRFZZ	FIRE
$h_L \equiv$ ZHLZZ	LAYR	$\pi \equiv$ PI	DATA BLOCK
$h_o \equiv$ ZHOZZ	INPUT	$\sigma \equiv$ SIGMA	DATA BLOCK
$\kappa \equiv$ ZULZZ	ABSRB	$T_a \equiv$ ZKAZZ	INPUT
$W = L_y \equiv$ ZLRZY	INPUT	$T_W \equiv$ ZKWZZ(KR,1)	HEAT

Calculations	Remarks
$A_1 = xy$, $A_2 = (L - x)y$, $A_3 = x(W - y)$, $A_4 = (L - x)(W - y)$	Areas of quadrants into which room is divided by the center of the (target) object. As in RNLO.
$R_i = \sqrt{4A_i/\pi}$	Equivalent radii
$H = h_R - h_L - h_o$	Distance between object surface and layer interface.
$h'_L = h_L + H = h_R - h_o$	Distance between ceiling and object surface. Needed when object is immersed in the layer gases.

$$\tau_i = \begin{cases} 2\kappa\left[1+\left(\frac{h'_L}{R_i}\right)^2\right]\left[h'_L + R_i - \sqrt{R_i^2 + h'^2_L}\right], & H \le 0 \\ 2\kappa\left[1+\left(\frac{H+h_L}{R_i}\right)^2\right]\left[h_L + \sqrt{R_i^2 + H^2} - \sqrt{R_i^2 + (H+h_L)^2}\right], & H > 0 \end{cases}$$

Mean opacity of layer

$$\omega = \begin{cases} 1 & H \le 0 \\ \frac{1}{4}\sum_{i=1}^{4}\left(1 + \frac{\pi H^2}{4A_i}\right)^{-1} & H > 0 \end{cases}$$

View factor of extended ceiling, at object KO.

$$\omega e^{-\tau} = \begin{cases} \frac{1}{4}\sum_{i=1}^{4} e^{-\tau_i} & H \le 0 \\ \frac{1}{4}\sum_{i=1}^{4} \frac{e^{-\tau_i}}{1+\frac{\pi H^2}{4A_i}} & H > 0 \end{cases}$$

Calculations (cont'd)	Remarks
$\phi = \omega e^{-\tau}\, \sigma T_W^4 + (1-\omega)\sigma T_a^{\,4}$	
$\phi_{WO} = \phi$	If object is not burning.
$\phi_{WO} = \phi \dfrac{e^{-\tau'} + \sin\psi}{1 + \sin\psi}$	If object is burning.
where $\tau' = R_f/\lambda$	Approximate opacity of flame.

RNPOØ1

Eqs. by H.E. Mitler, October 1977. Extended, August 1980.

In file R; called by RDNO.

Description. This subroutine calculates the radiant flux impinging normally on a horizontal target surface P, whose center is located a (horizontal) distance D from the center of (the base of) a solid radiating cone. This cone has semiapex angle ψ, base radius R_1, and is assumed to radiate as a uniform grey gas at temperature T. This is meant to approximate a flame (of base radius R_1). It is located on a fuel slab which extends a distance S_o beyond the flame center, in the direction of the target (see figs. 29-32). For points external to the fire, the flame is assumed to appear truncated by the sooty layer (figs. 29 and 30). If the target lies above the fire base, the "visible" part of the cone is truncated from below, as well (fig. 29). If the target lies below the fire base, the width of the fuel slab comes into account explicitly (figs. 30 and 32), and the geometry becomes much more complicated. When D = 0 -- that is, the target is the fire base itself -- then the entire flame is assumed to radiate down to the base, even when there is an "intersecting" sooty layer.

This is a complete revision of RNPO as it appears on p. 50-51 of CFC III (the derivation of those equations is given in TR34, p. 24-29). It is also an extension of RNPO as it appears in Mark IV; the principal extensions are: 1. to include the case shown in fig. 30 -- that is, where the base of the fire lies above the target. 2. The target may itself be burning. 3. We have also included the flame-flame interaction, via the (new) sub-subroutine RNFF. The mean flux over the fire base (from the flame) was taken to be 2/3 ds of the value at the center, in Mark IV. A more precise expression is used here. Finally, the flame height is made to decrease near fuel exhaustion. The derivation of the equations used in Mark IV is given in TR 34, p. 29-32.

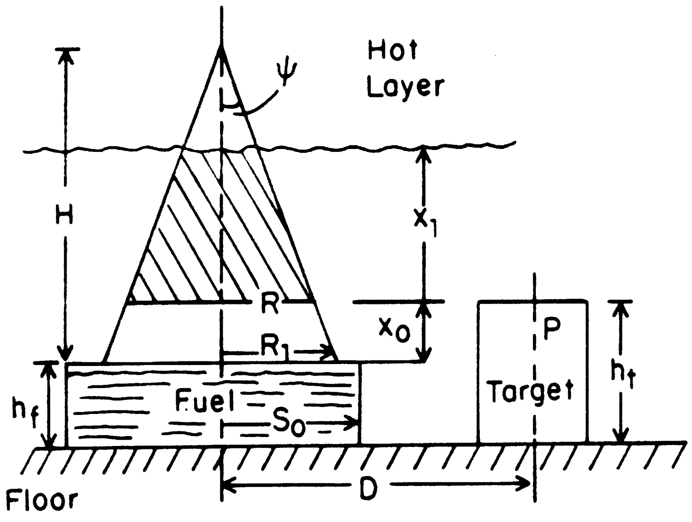


Fig. 29. First possible configuration, for an external target. The shaded part of the cone is the part which radiates to the point P.

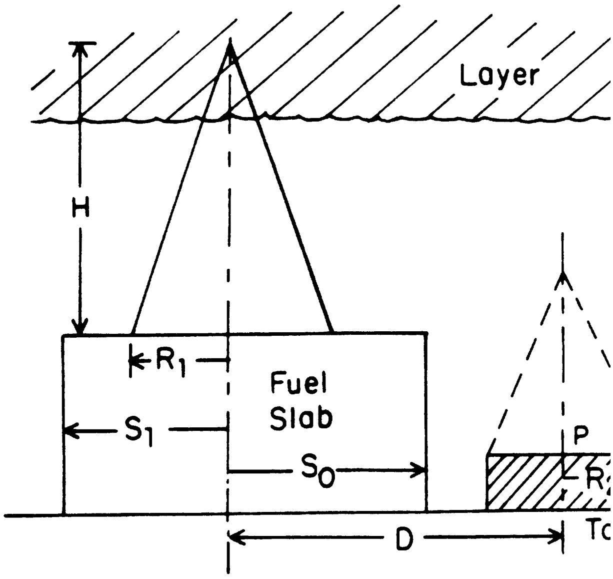


Fig. 30. The other possible configuration: the base of the fire lies above the target surface. The (original) fuel slab has length $S_0 + S_1$, and shades the target, at least in part. The cone above the target is dashed, because the target may or may not be burning.

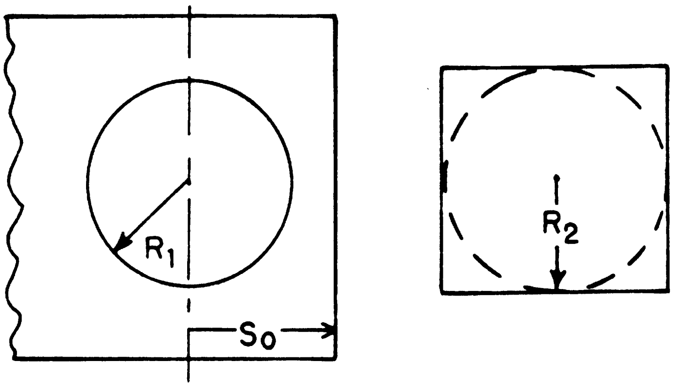


Fig. 31. Top view of the configuration shown in Fig. 30.

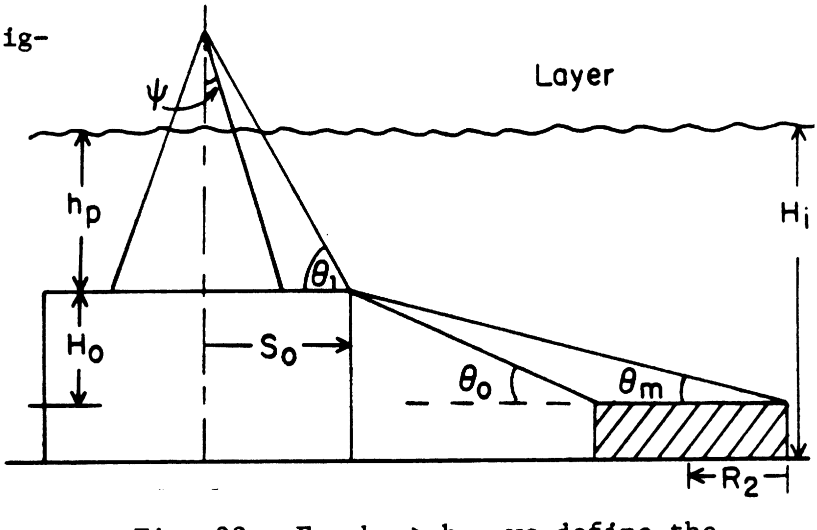


Fig. 32. For $h_f > h_t$, we define the angles shown here (see text).

Output

$\sum_f \Phi_{fo} \equiv$ FQPOR(KO)

Input required from COMMON	Found or Computed in	Remarks
$x_t \equiv$ ZXOZZ(KO)	INPUT	KO is the target object
$y_t \equiv$ ZYOZZ(KO)	INPUT	KO is the target object
$x_o \equiv$ ZXOZZ(KO2)	INPUT	KO2 is the flaming object
$y_o \equiv$ ZYOZZ(KO2)	INPUT	KO2 is the flaming object
$R_o \equiv$ ZRFZI(KO2)	INPUT	Radius of cylindrical fuel slab. If slab is rectangular, with distance S_o between flame center and edge of slab, use equivalent radius $R_o=2S_o/\sqrt{\pi}$. See figs. 29-32.
$R_2 \equiv$ ZRFZI(KO)	INPUT	Width (or radius) of target. If it is burning, R_2 is the fire radius.

Input required from COMMON	Found or Computed in	Input required from COMMON	Found or Computed in
$m_f \equiv$ ZMOZZ(KO2)	FIRE	$h_p \equiv$ ZHPZZ(KO2)	PLUM
$R_1 \equiv$ ZRFZZ(KO2)	FIRE	$\Phi_{ff} \equiv$ FQPP	RNFF
$\tan\psi \equiv$ TPSI(KO2)	FIRE	$k \equiv$ ZUFZZ	DATA
$\dot{m}_f \equiv$ TMOZZ(KO2)	FIRE	$T \equiv$ ZKFZZ(KO2)	DATA
$h_t \equiv$ ZHOZZ(KO)	INPUT	$\pi \equiv$ PI	DATA
$h_f \equiv$ ZHOZZ(KO2)	INPUT	$\sigma \equiv$ SIGMA	DATA

Calculations	Remarks
For each flaming object KO2, we calculate the flux to the target KO, as follows:	
$H_i = h_R - h_L$	Height of interface above the floor.
h_t or $h_f > H_i \Longrightarrow \Phi_{FO} = 0$	
$D = \sqrt{(x_t - x_o)^2 + (y_t - y_o)^2}$	Distance between center of target and center of flame.
$S_o = R_o\sqrt{\pi}/2$	Equivalent center-edge distance, for rectangular slab.

If $m_f \geq 0.2$ kg,

$H = R_1 \cot \psi$ — Height of flame (i.e. of cone)

When burnout of the fuel nears, we reduce the height of the flame appropriately. t_c is the first timestep at which we find $m_f < 0.2$ kg; then if $m_f < 0.2$ kg, define $\alpha_o \equiv H/\sqrt{|\dot{m}_f(t_c)|}$,

and thereafter $H(t) = \alpha_o\sqrt{|\dot{m}_f(t)|}$, $t > t_c$.

$x_o = h_t - h_f$ — Height of target above fire base

Case 1. $x_o \geq H$ — Target surface lies above tip of flame, so that it gets no flux.

$\Longrightarrow \phi_{FO} = 0$

Case 2. $x_o < 0$ — This corresponds to the configuration shown in figs. 30 and 32. We then have three subcases:

Define $H_o \equiv -x_o = h_f - h_t$

Case 2a. $\theta_1 \geq \theta_o$

i.e. $\tan \theta_1 \geq \tan \theta_o$, — The entire target surface is irradiated by flux from some part of the flame.

or $\dfrac{H}{S_o} \geq \dfrac{H_o}{D - S_o - R_2}$

$$L_{eff} = \frac{1}{3D^2(D-S_o)}\,\frac{H_o \tan^2\psi}{[(D-S_o)^2+H_o{}^2]^{3/2}}\left([H(D-S_o) - H_oS_o]^3 - (D-S_o)^3\Delta V'\right)$$

where $\Delta V' = \max[0,(H - h_p)^3]$.

Also, $$L = 0.77\left[R_1 - \frac{S_oH_o \tan\psi}{D-S_o}\right]$$

and $$\phi_a = \sigma T^4 \frac{L_{eff}}{L}\left(1 - e^{-kL}\right)$$

$\tau' = kR_2$ — Approximate opacity of flame over target, when latter is burning.

$$\phi_{FO} = \phi(1,2) \equiv \begin{cases} \phi_a e^{-\tau'} & \text{Target is flaming} \\ \phi_a & \text{Not flaming} \end{cases}$$

ϕ_{FO} is the mean flux from flame (over object 1) to target object (object 2).

Case 2b: $\theta_m \leq \theta_1 < \theta_o$

Only part of the target surface "sees" any part of the flame.

$$\text{or } \frac{H_o}{D-S_o+R_2} \leq \frac{H}{S_o} < \frac{H_o}{D-S_o-R_2}$$

$$\phi_{FO} = \phi(1,2)\left[1 + \frac{\sin\phi\cos\phi - \phi}{\pi}\right]$$

Note that this takes on two values, depending on whether the target is burning, or not.

$$\text{where } \cos\phi = \frac{D - S_o - S_o H_o/H}{R_2}$$

N.B.: ϕ is in radians

Case 2c: $\theta_1 < \theta_m$ i.e., $\dfrac{H_o}{D-S_o+R_2} > \dfrac{H}{S_o}$

The target surface is entirely shaded by the fuel slab, or the fuel slab and the layer.

$$\text{or } \frac{H}{D-S_o+R_2} > \frac{h_p}{S_o}$$

$$\phi_{FO} = 0$$

Case 3: $0 \leq x_o < H$

Figure 29 applies.

$D > R_1 \implies x_1 = \min(h_p, H) - x_o$

$D \leq R_1 \implies x_1 = H - x_o$

Target lies within flame radius; in this case we assume that the entire cone radiates down -- whether or not the layer intersects the flame.

$x_1 \leq 0 \implies \Phi_{FO} = 0$

No radiative flux at all, if target surface is in the layer.

$x_1 > 0$: $R - R_1 - x_o \tan\psi$

Base radius of visible part of cone

If $R < 0$, $\phi_{FO} = 0$ (for $D > 0$). If $D = 0$, then set $L = 0$ and go on to find $\overline{\phi}_{FO}$ as below.

$\xi = D/R$

$$\left.\begin{array}{l} \xi \geq 1 \implies L = D\left[1 + \dfrac{0.2787}{0.4349 - \xi^2}\right] \\ \\ \xi < 1 \implies L = 0.5068\ D\ \xi^{2.8254} \end{array}\right\}$$

L is an effective distance (a modified D, essentially)

$$s_o = \sqrt{L^2 + R^2}$$

$$s_1 = \sqrt{L^2 + x_1^{\ 2} + (R - x_1 \tan\psi)^2}$$

$$= 2k\left\{\sqrt{L^2 + x_1^{\ 2}} - L - (s_1 - s_0)\cos^2\psi + R\sin\psi\cos^2\psi \ln\left(\frac{s_o - R\sin\psi}{s_1 + x_1\sec\psi - R\sin\psi}\right)\right\}$$

Calculations (cont'd)

$\tau = 1.27324\ kR\left[1 + 0.84(\frac{R}{D})^2\right]$ Effective mean opacity (non-dimensional optical depth) of cone.

$\tau = \max(\tau, 10^{-6})$ To avoid "overflow" problems in the computer.

Then for $L > 0$, $\phi_{FO} = \Lambda\sigma T^4\left(\frac{1-e^{-\tau}}{\tau}\right)\begin{cases}1 & \text{non-flaming target} \\ e^{-\tau'} & \text{flaming target}\end{cases}$

while for $L = 0$, (flame radiating to its own fire base)

$$\overline{\phi}_{FO} = \sigma T^4(1 - e^{-0.7755\Lambda})$$

(For the center of the base, the factor 0.7755 would not appear; it is there in order to give the mean flux over the entire base). Finally,

$\Phi_{FO} = \phi_{FO} + \phi_{FF}$ Whatever flux it has received directly, it may receive some more from the fire, via the flame-flame interaction, if the target is also burning.

We then sum over all the flames:

$$\rightarrow \sum_f \Phi_{FO}$$

RNFFØ2

Eqs. by H.E. Mitler, July 1980.

In file R; called by RNPO.

Discussion. This sub-subroutine calculates the energy absorbed by a flame due to radiation from another flame which may be shaded by a fuel slab. Each flame is modeled by a homogeneous gray gas in the form of a cone of semiapex angle ψ_i. The slab upon which one flame rests lies a height H_o above the base of the other flame, and is opaque; its leading edge is orthogonal to the line connecting the (parallel) axes of the cones, as shown in fig. 33 below. As shown in fig. 34, one or both of the flames may be interrupted by the hot, sooty gas layer, assumed to be entirely opaque for the present calculation. Once the energy absorption rate is found, the flux to the surface due to this extra energy is calculated.

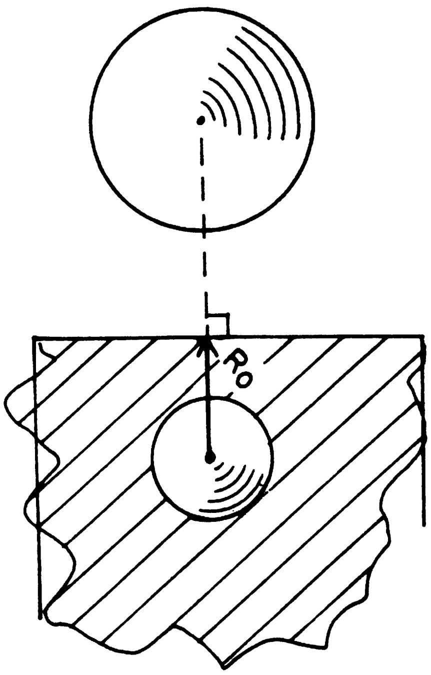


Fig. 33. Top view of cones modeling the fires, with #1 sitting on its pedestal.

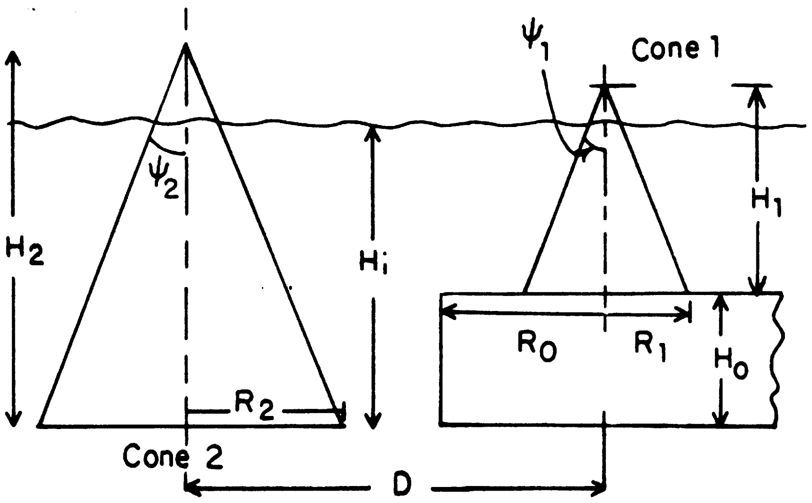


Fig. 34. Side view of configuration shown in fig. 33.

Needed input	Source	Remarks
R_1, R_2	FIRE	Radii of flame bases, ZRFZZ(i)
R_O	INPUT	Distance from cone 1 to edge of slab
ψ_1, ψ_2	INPUT	Semiapex angles of cones, PSI(i)
$H_i = h_p(2)$	PLUM	ZHPZZ(2)
H_O	INPUT	Height of base 1 above base 2
k_1, k_2	INPUT	ZUFZZ(i), i = 1,2
T_1, T_2	INPUT	ZKFZZ(i), i = 1,2
$(x,y)_1$; $(x,y)_2$	INPUT	Positions of objects 1 and 2 (i.e., flame centers).

Output

ϕ_{ij}	The part of the flux to fire base j due to flame i (i = 1,2; j = 2,1).

Calculations | Remarks

$$G = 2k_1k_2\sigma T_1^4/D^2$$

$$H_u = \begin{cases} H_2 - H_o & H_2 > H_o \\ 0 & H_2 \leqslant H_o \end{cases}$$

Amount by which height of cone 2 exceeds height of slab on which cone 1 stands

$$H_c = \frac{H_oR_o}{D - R_o + R_2}$$

$$\frac{V_L}{A_L} = \begin{cases} \frac{1}{3}\,\dfrac{H_2^3 - H_u^3}{H_2^2(1 + \csc\psi_2) + H_u^2(1 - \csc\psi_2)} & H_1 > H_c \\ \frac{1}{3}\,\dfrac{[H_u + (D-R_o)H_1/R_o]^3 - H_u^3}{[H_u + (D-R_o)H_1/R_o]^2(1 + \csc\psi_2) + H_u^2(1-\csc\psi_2)}\,, & H_1 < H_c \end{cases}.$$

V_L/A_L is the ratio of absorbing volume of lower part of cone 2, to its bounding area. Four times this quantity is (approximately) the mean beam length. Note that if $H_i < H_o$, there *is* no interchange of fluxes.

$$\frac{V_u}{A_u} = \frac{H_u}{3(1 + \csc\psi_2)}$$

Same remarks apply as for V_L/A_L, but for the *upper* part of the cone. We neglect the possibility that the hot layer truncates the cone, for simplicity

$$v_1 = \frac{V_1}{\pi} = \begin{cases} \frac{1}{3}\,R_1^2H_1 & H_i > H_1 + H_o \\ \frac{1}{3}\,R_1^2H_1 - \frac{1}{3}\,(H_o + H_1 - H_i)^3\tan^2\psi_1 & H_o < H_i < H_1 + H_o \\ 0 & H_i < H_o \end{cases}$$

V_1 is the part of cone 1 which is not obscured by the hot layer.

$$A_{pu} = \begin{cases} A_u = 2D^2\left(\sin^{-1}\dfrac{R_u}{D}\right)\cdot\left[\dfrac{\sqrt{H_u^2+D^2}-D}{H_u}\right] & H_i > H_2 \\[2ex] A_u - 2D^2\sin^{-1}\left(\dfrac{R_2 - H_i\tan\psi_2}{D}\right)\cdot\left[\dfrac{\sqrt{(H_2-H_i)^2+D^2}-D}{H_2-H_i}\right] & H_2 > H_i > H_0 \\[2ex] 0 & H_0 \geqslant H_i \end{cases}$$

A_{pu} is the (approximate) projected area of the upper section of cone 2.

$$A_{pL} = \begin{cases} 2D^2\sin^{-1}\dfrac{R_2}{D}\left[\dfrac{\sqrt{H_2^2+D^2}-D}{H_2}\right] - A_u & H_1 > H_c \\[2ex] (D-R_0)\,[2H_u + (D-R_0)H_1/R_0]\,\dfrac{H_1}{R_0}\tan\psi_2 & H_1 \leqslant H_c \end{cases}$$

A_{pL} is the (approximate) projected area of the visible part of the lower section of cone 2.

$$2M = 1 + \left[1 - \frac{R_0H_0}{H_1(D-R_0-R_2)}\right]^3$$

M is (roughly) the mean fraction of the emitting cone, seen by the lower part of the absorbing cone.

$$\dot{\varepsilon}_u \cong 2\frac{V_u}{A_u}A_{pu}$$

$$\dot{\varepsilon}_L \cong 2\frac{V_L}{A_L}A_{pL}$$

$\dot{E}_u = \dot{\varepsilon}_u G v_1$ is the power absorbed by the upper part of cone 2. $\dot{E}_L$, that by the lower part, if not shadowed.

$$\dot{E}_{12} = Gv_1(\dot{\varepsilon}_u + \dot{\varepsilon}_L M)$$

Power absorbed by cone 2, from radiation emitted by cone 1.

$$\dot{E}_{21} = \dot{E}_{12}\left(\frac{T_2}{T_1}\right)^4\left(\frac{1+\sin\psi_2}{1+\sin\psi_1}\right)\left(\frac{\cos\psi_1}{\cos\psi_2}\right)$$

Power absorbed by cone 1, from radiation emitted by cone 2.

$$\phi_{12} = \dot{E}_{12}/\pi R_2^2(1+\csc\psi_2)$$

Flux to base of fire 2, due to fire 1, *via* flame-flame interaction.

$\phi_{21} = \dot{E}_{21}/\pi R_1^2(1 + \csc\psi_1)$	Flux to base of fire 1, due to fire 2, *via* flame-flame interaction

RDNP

Eqs. by H.E. Mitler, May 1978

In file R; called by CALS

Description. This subroutine calculates the radiative heat loss from each flame. Although we still use a uniform, homogeneous cone of gas as a model for the flame, the expression has been cast in a form such that if we prefer to use a conical *shell* instead, we can do so. (LIMITS has been inserted, of course.)

Just as in RNPO, when m_f falls to 200 gm, the flame height is reduced. (We do not take into account the possibility of an object having an initial mass less than 200 gm.) The equations are derived in TR34, pp. 23-24.

Output

$\dot{E}_{PR}$ = TEPZR(KO)

Input required from COMMON	Found or Computed in
σ ≡ SIGMA	DATA
R ≡ ZRFZZ(KO)	FIRE
$\dot{E}_F$ ≡ TEOZZ(KO)	FIRE
π ≡ PI	DATA
$\tan\psi$ ≡ TPSI	FIRE
α ≡ ALEPH	DATA

Input	From
T ≡ ZKFZZ	DATA
κ ≡ ZUFZZ(KO)	DATA
$\dot{m}_f$ ≡ TMOZZ(KO)	FIRE
m_f ≡ ZMOZZ(KO)	FIRE

Remarks

If a conical *shell* is desired, then αR is the radius of the nonluminous inner cone--i.e., $(1-\alpha)R$ is the conical shell's thickness. $\alpha = 0$ yields the solid cone, therefore; $\alpha = 0$ is the default value.

Calculations	Discussion
$\alpha_0 = R \cot \psi / \sqrt{\lvert \dot{m}_f(t_c) \rvert}$	When m_f falls to 200 gm, the flame height is reduced. t_c is t at the first time step such that $m_f \leqslant 200$ gm.
$H(t) = \alpha_0 \sqrt{\lvert \dot{m}_f(t) \rvert}\ ,\quad t > t_c$	
$\cos \psi = H/\sqrt{R^2 + H^2}\ ,\quad t > t_c$	
$A = \pi R^2 [(1+\csc\psi)(1+\alpha^2) - 2\alpha^2]$	Area of cone (or conical shell)
$V = \frac{\pi}{3} R^3 (1-\alpha^3) \cot \psi$	Volume of cone (or conical shell)
$\dot{E}^*_{PR} = A\sigma T^4 [1 - e^{-4\kappa V/A}]$	Total radiative emission rate from flame
$\dot{E}_{max} = -0.43\ \dot{E}_f$	At late stages, the flame may be limited by lack of oxygen. In that case, radiation loss must be limited also.
$\dot{E}_{PR} = \min(\dot{E}_{max}, \dot{E}^*_{PR})$	

RDNW

Eqs. by H.E. Mitler, June 1978

In file R; called by CALS

Description. This subroutine calculates the radiative heat transfer to the portion of the walls and ceiling contiguous with the hot layer, from the layer and the flame(s). The layer is approximated by a cylinder, with each flame axially located and itself further approximated by a point source at its centroid. This is essentially the same as in CFCIV. The equations are derived in TR34, pp. 36-37.

Output	Definition
$\sum_f \phi_{FW} \equiv$ FQPWR(KO)	Mean radiative flux incident on extended ceiling, from flames. (In w/m^2).

$\phi_{LW} \equiv$ FQLWR(KR) — Mean flux incident on extended ceiling from the hot layer (in room KR)

Input required from COMMON	Found or calculated in
$\dot{E} \equiv$ TEPZR(KO)	RDNP
$H_p \equiv$ ZHPZZ(KO)	PLUM
$R \equiv$ ZRFZZ(KO)	FIRE
$L_y \equiv$ ZLRZY(KR)	INPUT
$L_x \equiv$ ZLRZX(KR)	INPUT

Input	From
$h \equiv$ ZHLZZ(KR)	LAYR
$\kappa \equiv$ ZULZZ(KR)	ABSRB
$T_L \equiv$ ZKLZZ(KR)	LAYR
$\pi \equiv$ PI	DATA

Calculations	Remarks
For each flame above an object KO:	
$\theta = \min[h/2, 1/\kappa]$	
$H = H_p - R$	
If $H \geqslant 0$,	
$\theta_1 = \min(H,\theta)$	
$r = \sqrt{L_x L_y/\pi}$	Equivalent ceiling radius
$\theta_0 = \tan^{-1} \dfrac{r}{H+\theta_1}$	Mean acceptance angle for flame (by the layer)

$$\Rightarrow \sec\theta_0 = \sqrt{1 + \left(\frac{r}{H+\theta_1}\right)^2}$$

and

$$\dot{E}_{FC} = \frac{\dot{E}}{2}\left\{\cos\theta_0\left[1 - \frac{e^{-\kappa h\sec\theta_0}}{1+\kappa h\sec\theta_0}\right] + \frac{e^{-\kappa h}}{1+\kappa h} - \frac{H}{\sqrt{r^2+H^2}}\right\}$$

= radiative power impinging on walls and ceiling, from fire(s).

If $H < 0$,

$$\theta_2 = \tan^{-1}|H/r| \Rightarrow \csc\theta_2 = \sqrt{1+(r/H)^2}$$

$$\dot{E}_{FC} = \frac{\dot{E}}{2}\left\{\frac{e^{-\kappa h}}{1+\kappa h} + \sin\theta_2 \frac{e^{-\kappa h\csc\theta_2}}{1+\kappa h\csc\theta_2}\right\}$$

$A = L_x L_y + 2(L_x + L_y)h$	Area of walls and ceiling
$\phi_{FW} = \dot{E}_{FC}/A$	Summing over all flames, we get $\sum_f \phi_{FW}$
$V = hL_x L_y$	Volume of layer
$\zeta = \dfrac{4\kappa V}{A + L_x L_y}$	(Approximate) mean opacity of layer
$\varepsilon = 1 - \exp\left(\dfrac{-\zeta}{1 + 0.18\zeta}\right)$	Emittance of layer
$\phi_{LW} = \sigma T_L^4$	

RDNL

Equations by H. Mitler, May 1978
Modified 7/78, 10/80.
In file R; called by CALS

Description. This subroutine finds the net power gain of the layer, *via* radiation. It is an improvement over the calculation shown on page 55 of CFC III in that (a) it uses the emissivity of the layer, calculated in ABSRB, (b) it takes the exposed vent areas into account. (c) A variable beam length approximation is used for the emittance of the layer.
(d) The hot and cold walls and the fire(s) are explicitly taken into account. This differs from the version used in Mark IV in correcting a small error, and in using the LIMITS test. The equations are derived on pp. 37-39 of TR34.

Output

$\dot{Q} \equiv$ TELZR(KR)

Input required from COMMON	Found or calculated in
$T_a \equiv$ ZKAZZ	DATA
$T_L \equiv$ ZKLZZ(KR)	LAYR
$T_W \equiv$ ZKWZZ(KW,1)	TMPW
$h_L \equiv$ ZHLZZ(KR)	LAYR
$h_p \equiv$ ZHPZZ(KO)	PLUM
$R \equiv$ ZRFZZ(KO)	FIRE
$W = L_y \equiv$ ZLRZY(KR)	INPUT
$L = L_x \equiv$ ZLRZX(KR)	INPUT

Input	From
$\sigma \equiv$ SIGMA	DATA
$\dot{E} \equiv$ TEPZR(KO)	RADP
$\sum_f \phi_{FW} \equiv$ FQPWR(KW,1)	RADW
$\kappa \equiv$ ZULZZ(KR)	ABSRB
$B \equiv$ ZBVZZ(KV)	INPUT
$h_V \equiv$ ZHVZZ(KV)	INPUT
$h_T \equiv$ ZHTZZ(KV)	INPUT

Calculations

$h_1 = h_L - h_T$

$h_2 = \max(0, h_1)$

$h_3 = \min(h_V, h_2)$

$A_V(KV) = h_3 B$ — Area of vent KV which is covered by the layer

$A_V = \sum_{KV} A_V(KV)$ — Total vent area covered

$V = WLH_L$ — Volume of hot layer

$A = 2(WL + Lh_L + Wh_L)$ — Bounding area of hot layer

$\zeta = 4\kappa V/A$

$\bar{\varepsilon} = 1 - \exp\left(\frac{-\zeta}{1 + 0.18\zeta}\right)$ — Mean emittance of layer

$$\dot{Q}_{LR} = -\bar{\varepsilon}\sigma[AT_L^4 - (WL+A_V)T_a^4 - (A-WL-A_V)T_W^4]$$

Power gain of hot layer, taking only the radiative exchange with "walls" into account.

Next, for each flame we do the following:

$H = h_p - R$ — Distance from fire centroid to hot layer

$r = \sqrt{WL/\pi}$	Equivalent radius of ceiling
$\dot{E}_f = \frac{\dot{E}}{2}\left[1 - \frac{H}{\sqrt{r^2 + H^2}}\right]$	The part of the radiative output of the flame which is incident on the hot layer
$\dot{E}_{FC} = (A - WL)\Sigma_f \phi_{fW}$	Flame radiation to extended ceiling, or going out the vents
$\dot{E}_a = \Sigma_f \dot{E}_f - \dot{E}_{FC}$	The part of the flame radiation absorbed by the layer
$\dot{Q} = \dot{Q}_{LR} + \dot{E}_a$	Total radiative power gain of layer.

CNVWØ1

Equations by H. Emmons, June 1977.

In file C; called by CALS.

Description. This subroutine computes the convective flux to the walls of a room; i.e., the convective heat transfer rate per unit area, from a hot layer to the walls and ceiling of a room. The hot layer is assumed to have the same temperature throughout and likewise the surface of the walls and ceiling (referred to as "walls") are assumed to have a uniform temperature where they are in contact with the hot layer. This is quite unchanged from the version in CFC III (listed there on p. 37). The equations are developed on p. 48-49 of T.R. 34.

Output $\dot{q}''_{LW} \equiv$ FQLWD(KW,1), $\dot{q}''_{AW} \equiv$ FQLWD(KW,2)

Input required from COMMON	Found or calculated in	Input	From
$T_i \equiv$ ZKWZZ(KW,1)	TMPW	$T_A \equiv$ ZKAZZ	DATA
$T_e \equiv$ ZKWZZ(KW,2)	TMPW	$b \equiv$ ZOAZZ	DATA
$T_L \equiv$ ZKLZZ(KR)	LAYR	$a \equiv$ ZOAZM	DATA

Calculations	Remarks
$h_e = b$	Outside heat transfer coefficient
$h_i = \mathrm{Min}\left(a, b + (a - b)\frac{(T_L - T_A)}{100}\right)$	Inside heat transfer coefficient
$\dot{q}''_{LW} = h_i(T_L - T_i)$	Heat flux to inside wall surface
$\dot{q}''_{AW} = h_e(T_A - T_e)$	Heat flux to outside wall surface

CNVL

Equations by H. Emmons, June 1977.
Modified by M. Gilson, August 1978.
Modified by H. Mitler, Sept. 1980.
In file C; called by CALS.

Description. This subroutine finds the convective heat loss rate from the hot layer (to the extended ceiling). The hot layer is assumed to be at a uniform temperature as are the surfaces of the walls and ceiling in contact with the hot layer. There is only one hot layer in each room. The routine also assumes only one "wall" per room, and that side two of the "wall" is in contact with the outside -- i.e., with quiescent ambient air. This version differs from CNVLØ1 as given on page 59 of CFC III in three ways: (1) the sign error in the definition has been removed. (2) The parts of the vent areas contiguous to the layer have been taken into account. (3) It is clear that the expression formerly used for the rate of heat injection into newly covered parts of the wall was overestimated. Calculations show that it is of the right order of magnitude, however. Therefore we take just half that amount. The equations are derived on pp. 49-51 of TR 34.

Output

$\dot{E}_{LD}$ ≡ TELZD

Input required from COMMON	Found or calculated in	Input	From
h_t ≡ ZHTZZ(KV)	INPUT	h_L ≡ ZHLZZ	LAYR
h_v ≡ ZHVZZ(KV)	INPUT	$\dot{E}_L$ ≡ TELZZ	LAYR
B_v ≡ ZBVZZ(KV)	INPUT	E_L ≡ ZELZZ	LAYR
$L = L_x$ ≡ ZLRZX	INPUT	$\dot{q}''_{LW}$ ≡ FQLWD(KW,1)	CNVW
$B = L_y$ ≡ ZLRZY	INPUT	$\dot{q}''_{AW}$ ≡ FQLWD(KW,2)	CNVW

Calculations	Remarks
$A_W = 2(B + L)h_L$	Area of heated (vertical) wall

Calculations (cont'd)	Remarks
$D_1 = h_L - h_t(KV)$	
$D_2 = \max(0,D_1)$	
$D_3 = \min(D_2,h_v)$	
$A_v = B_v D_3$	Area of vent KV which is covered by the layer
$A = AW + BL - \sum\limits_v A_v$	Area of heated part of wall and ceiling which is not covered by vents
$\dot{h}_L = h_L \dot{E}_L / E_L$	Rate of descent of layer
$q'' = \int_0^t (\dot{q}''_{LW} + \dot{q}''_{AW})dt$	Convective energy stored in unit area of hot wall
$\dot{E}_{LW} = \begin{cases} \dot{q}''_{LW}A + q''(B + L - \frac{1}{2}\sum\limits_v{}' B_v)\dot{h}_L & \text{if } \dot{E}_L > 0 \\ \dot{q}''_{LW}A & \text{if } \dot{E}_L \le 0 \end{cases}$	Energy loss rate by convection from the layer to wall

The prime on the sum over v indicates that we sum only over those vents which are cut by the layer.

$\dot{E}_{LO} = -\dot{E}_{LW}$	Rate of change of layer energy due to convective heating (or cooling) of the walls/ceiling.

TMPWØ1

Equations by H. Emmons, June 1977.

Modified by H. Mitler, Sept. 1980.

This subroutine appears in file T, and is called by CALS.

Discussion. A wall of thickness θ is exposed to radiative and convective heat fluxes on its front and back surfaces. The time-dependent temperature profile through the wall is calculated numerically by cutting the wall into slabs (parallel to its front surface), and solving the one-dimensional heat-diffusion equation by differences. This subroutine is an improvement over subroutines HEAT(CFC IV) and TEMPØ1(CFC III). HEAT improved on TEMPØ1 by taking radiation from the cold walls into account, as well as reradiation from the hot wall/ceiling itself. TMPØ1 improves on HEAT principally in using a better expression for the absorptivity of the layer, and restructuring the program more efficiently; it also includes LIMITS, etc.

See Technical Report 37, p. 51 ff, for a derivation of the equations.

Output for wall KW:

$T_i \equiv$ ZKWZZ(KW,1) $\qquad T_e \equiv$ ZKWZZ(KW,2)

Input required from COMMON	Found or calculated in		Input	From
$\dot{q}''_{LWD} \equiv$ FQLWD(KW,1)	CNVW		$k \equiv$ ZJWZZ(KW)	INPUT
$\dot{q}''_{LWR} \equiv$ FQLWR(KW,1)	RDNW		[$\alpha \equiv$ ZGWZZ(KW)	INPUT]
$\dot{q}''_{PWR} \equiv$ FQPWR(KW,1)	RDNW	or	$c \equiv$ ZCWZZ(KW)	INPUT
$\dot{q}''_{AWD} \equiv$ FQLWD(KW,2)	CNVW	and	$\rho \equiv$ VMWZZ(KW)	INPUT
$\dot{q}''_{AWR} \equiv$ FQLWR(KW,2)	RDNW		$\kappa \equiv$ ZULZZ	ABSRB
$\dot{q}''_{PR} \equiv$ FQPWR(KW,2)	RDNW		$\sigma \equiv$ SIGMA	DATA
$\Delta t \equiv$ DT	MAIN		$h_L \equiv$ ZHLZZ	LAYR
$\theta \equiv$ ZNWZZ(KW)	INPUT		$T_i \equiv$ ZKWZZ(KW,1)	TMPW
$T_i(x)$	INPUT		Initial temperature distribution through the wall (generally taken to be constant, at T_a).	

Calculations	Remarks
$\delta x_n = \sqrt{2\alpha\Delta t}$	Smallest permissible space increment (calculated for the largest time increment to be used).
$N = 1 + \left[\frac{\theta}{\delta x_n}\right]$ (integer part of)	N is number of points where temperature is calculated. n = 1 on inside surface, n = N on outside surface.

$$\left.\begin{array}{ll} \delta x = \frac{\theta}{N-1} & 1 < N \leq 20 \\ \delta x = \frac{\theta}{19} & N > 20 \\ \delta x = \theta & N = 1 \end{array}\right\}$$

Space increment actually used

Calculations (cont'd)	Remarks
$a = \frac{2\alpha}{\delta x^2}$ $b = \frac{\delta x}{k}$	Computation parameters.

Set $T_n(0) = T_i((n-1)\delta x) \qquad n = 1 \rightarrow N$

The calculation of the temperature distribution at time $t + \Delta t$ given the distribution at time t:

$\dot{q}''_1 = \dot{q}''_{LWD} + \dot{q}''_{LWR} + \dot{q}''_{PWR}$	Heat flux to side 1 (inside).
$\dot{q}''_N = \dot{q}''_{AWD} + \dot{q}''_{AWR} + \dot{q}''_{PR}$	Heat flux to side N (outside).
$\phi_1 = \dot{q}''_1 - \sigma T_i^4 + e^{-\kappa h_L} \sigma T_a^4$	Net flux to inside of wall after reradiation
$[\phi_N = \dot{q}''_N + \sigma T_a^4 - \sigma T_e{}^4$	Net flux to outside of wall.]*
$T_1(t+\Delta t) =$ $T_1(t)(1 - a\Delta t) + a\Delta t(T_2(t) + b\phi_1)$	Temperature on inside surface.
$T_n(t+\Delta t) =$ $'T_n(t)(1 - a\Delta t) + \frac{a\Delta t}{2}(T_{n-1}(t) + T_{n+1}(t))$	Interior temperature.
$T_N(t+\Delta t) =$ $T_N(t)(1 - a\Delta t) + a\Delta t(T_{N-1}(t) + b\phi_N)$	Temperature on outside surface.
$T_i = T_1$	Temperature of the inside surface.
$T_e = T_N$	Temperature of the outside surface.

For marinite walls in the 1977 Home Fire Project test fires, we had

$T_i = 300^o K$	$k = .134 \frac{w}{mK}$
$\theta = 1$ inch $= .0254$ m	$\alpha = 1.577 \times 10^{-7} \frac{m^2}{sec}$

* Through an oversight, Mark 5 takes $\phi_N = \dot{q}''_N$ -- i.e., reradiation is neglected.

TMPO$\emptyset$1

Eqs. by H.E. Mitler, July 1978; revised Sept. 1980.
In file T; called by CALS.

<u>Discussion</u>. This subroutine finds the temperature of the surface of an object exposed to heating fluxes. It is an improvement on subroutine IGNT (which was used in CFC IV, which itself replaced TEMP12, in CFC III). The radiation considered is from the walls, the layer, and any flames, to a horizontal surface. Heating or cooling by convection is also included. The shorcut method used here is approximate, and avoids the need for finding the temperature profile throughout the object. The object is assumed to be a semi-infinite slab. The derivation of the equations is found in Technical Report 34, p. 57-59. One of the improvements over IGNT is the $\dot{q}''_{WO}$ is now taken to be the <u>total</u> flux from the walls -- i.e., including that from the "cold" walls. That is obtained from the new subroutine, RNWO$\emptyset$2. Another improvement is the inclusion of heating by the layer, when the object is immersed in it. (Other changes improve the numerical stability, but that only appears in the program, rather than in the equations shown here).

<u>Output</u> $T_{\frown} \equiv$ ZKOZZ(KO)

Input required from COMMON	Calculated in subroutine	
$T_a \equiv$ ZKAZZ	INPUT	
$e_b \equiv$ EB	DATA BLOCK	
$h \equiv$ ZOAZZ	DATA BLOCK	
h_o, $h_1 \equiv$ ZOLZN, ZOLZM	DATA	
$\rho \equiv$ VMOZZ	INPUT	
$c \equiv$ ZCOZZ	INPUT	
$\kappa \equiv$ ZJOZZ	INPUT	
$\dot{q}''_{LO} \equiv$ FQLOR	RNLO	RDNO
$\sum_W \dot{q}''_{WO} \equiv$ FQWOR	RNWO	RDNO
$\sum_P \dot{q}''_{PO} \equiv$ FQPOR	RNPO	RDNO

Input	From
$h_L \equiv$ ZHLZZ	LAYR
$h_r \equiv$ ZHRZZ	INPUT
$h_s \equiv$ ZHOZZ	INPUT
$T_s \equiv$ ZKOZZP	TMPO$\emptyset$1
$\pi \equiv$ PI	DATA
$T_W \equiv$ ZKWZZ(1,1)	TMPW
$T_a \equiv$ ZKAZZ	DATA BLOCK
$L \equiv$ ZLRZY	INPUT
$W \equiv$ ZLRZX	INPUT

Calculations | Remarks

$\phi_a = \sum\limits_W \dot{q}''_{WO} + \dot{q}''_{LO} + \sum\limits_P \dot{q}''_{PO}$ — Radiative flux impinging on object KO from all sources

$\phi_b = e_b(\phi_a - \sigma T_s^{\,4})$ — Flux, reduced by reradiation

$$\phi = \begin{cases} \phi_b - h(T_s - T_a), & h_L + h_s < h_r \\ \phi_b + h'(T_L - T_S), & h_L + h_s \geq h_r \end{cases}$$

Net flux, after cooling or heating by convection as well.

$$\text{where } h' = \min\left[h_1, h_o + (h_1 - h_o)\left(\frac{T_L - T_a}{100}\right)\right]$$

$H = h_r - h_L - h_s$ — Distance between object and layer.

If a small numerical fluctuation makes this flux erroneously negative, near the start, use

$$\phi = e_b \sum_P \dot{q}''_{PO} + \frac{6(T_W - T_a)}{1 + \frac{\pi H^2}{LW}} - 16(T_s - T_a)$$

instead.

For $T_s < T_a + 5$,

$$p(t) = \phi(t) \Big/ \int_0^t \phi(t')dt'$$

Rate of growth of net heating flux

For $T_s > T_a + 5$,

$$p_{new} = \frac{1}{\Delta t} \ln\left[\frac{\phi(t)}{\phi(t-\Delta t)}\right]$$

However, if $\frac{\phi(t)}{\phi(t-\Delta t)} \leq 0$ or ≥ 10,

use $p_{new} = \frac{1}{\Delta t} \ln \left[\frac{T_s(t) - T_a}{T_s(t-\Delta t) - T_a}\right]$, instead.

$p' = 0.8\, p(t-\Delta t) + 0.2\, p_{new}$ — This moving average smooths out fluctuations.

$p(t) = \max(0, p')$ — Heat diffusion in the target is assumed to only cool the surface.

$$T_s(t) = T_a + \frac{T_s(t-\Delta t) - T_a}{\sqrt{1+9.65\, p(t)\Delta t}} + \sqrt{\frac{\Delta t}{\pi \rho c k}}\,[\phi(t-\Delta t) + \phi(t)]$$

TMPOØ2

Eqs. by H. Emmons, June 1977.

Modified by H. Mitler, Sept. 1980.

The subroutine name is a contraction of Temperature of Object.

In file T; called by CALS.

Discussion. A slab of thickness θ is exposed to radiative fluxes on one surface, and is convectively cooled when it lies in ambient air, or convectively heated if (when) it lies inside the hot layer. It is assumed that there is no net flux to the other surface. This surboutine calculates the time-dependent one-dimensional temperature profile through the object in the same way that the wall temperature is found in subroutine TMPWØ1. It is a generalization and improvement over TEMP11 in CFC III, principally in two ways: first, the calculation can be done for slabs with any values of the geometrical and physical properties. Second, convective heating is permitted as well as cooling.

Note that when a slab is very thick, the equations below will only permit quite thick elemental slabs, so that this calculation loses accuracy. In a case like that, however, the thermal wave will probably not penetrate the whole slab, and it is permissible to use a smaller effective thickness, yielding greater accuracy in that region. When the slab is thermally thin, it is not cut into slices at all.

The equations are derived in T.R. 34 on p. 51-54.

Output

T_s ≡ ZKOZZ(KO)

Input from COMMON	Found or calculated in		Input	From
$\sum_P \dot{q}''_{PO}$ ≡ FQPOR(KO)	RNPO	RDNO	T_a ≡ ZKAZZ	INPUT
$\dot{q}''_{LO}$ ≡ FQLOR(KO)	RNLO	RDNO	T_s ≡ ZKOZZ(KO)	TMPOØ2
$\sum_W \dot{q}''_{WO}$ ≡ FQWOR(KO)	RNWO	RDNO	T_L ≡ ZKLZZ(KR)	LAYR

Input (cont'd)	From
$\Delta t \equiv$ DT	MAIN
$\theta \equiv$ ZNOZZ(KO)	INPUT
$\alpha \equiv$ ZGOZZ(KO)	INPUT
$\kappa \equiv$ ZJOZZ(KO)	INPUT
$e_b \equiv$ EB(KO)	INPUT
$T_i(x)$	INPUT

Input	From
$h_o \equiv$ ZOAZN	DATA
$h_1 \equiv$ ZOLZM	DATA
$h_L \equiv$ ZHLZZ(KR)	LAYR
$h_s \equiv$ ZHOZZ(KO)	INPUT
$h_r \equiv$ ZHRZZ(KR)	INPUT

Calculations	Remarks
$\delta x_{min} = \sqrt{2\alpha\Delta t}$	Smallest permissible elemental slab thickness (calculated for the largest time increment to be used. Generally, this has been the initial Δt, 2 sec).
$M = 1 + [\theta/\delta x_{min}]$	The square bracket here means we take the integer part of the fraction.
$N' = \min(M,20)$ $N = \max(1,N')$	N is the number of points at which we calculate the temperature -- i.e., we have N-1 elemental slabs. n = 1 at the front (top) surface, n = N at the rear (bottom) surface, and we have at least one slice.
$\delta x = \theta/(N-1)$	Slab thickness -- i.e., grid distance.
$a = 2\alpha/(\delta x)^2$	
$b = \delta x/k$	
$T_n(t=0) = T_i[(n-1)\delta x]$, $n = (1,N)$	Initial temperature distribution
$\phi_a = \sum_P \dot{q}''_{PO} + \sum_W \dot{q}''_{WO} + \dot{q}''_{LO}$	Radiative flux to (top) surface
$\phi_b = e_b(\phi_a - \sigma T_s^4)$	Subtract reradiation
$h' = \min\left[h_1, h_o = (h_1 - h_o)\left(\frac{T_L - T_a}{100}\right)\right]$	Heat transfer coefficient of hot gases
$h_g = \begin{cases} 2h_o & h_L + h_s < h_r \\ h' & h_L + h_s \geq h_r \end{cases}$	Heat transfer coefficients of gases, as above. 10 w/m^2 deg is taken for air which is <u>not</u> quiescent.
$T_g = \begin{cases} T_a & h_L + h_s < h_r \\ T_L & h_L + h_s \geq h_r \end{cases}$	Gas temperature when object surface lies in cool gas (ambient air) and gas (layer), respectively.

$\phi = \phi_b - h_g(T_s - T_g)$ Net heating flux to surface

If a small numerical fluctuation makes this flux (erroneously) negative near the start, use

$$\phi = e_b \sum_p \dot{q}''_{PO} + \frac{6(T_W - T_a)}{1 + \frac{\pi H^2}{LW}} - 16(T_s - T_a),$$

instead.

$$T_1(t+\Delta t) = (1-a\Delta t)T_1 + a\Delta t(T_2+b\phi)$$

$$T_n(t+\Delta t) = (1-a\Delta t)T_n + \frac{a\Delta t}{2}(T_{n-1} + T_{n+1}) \qquad n = 2, \ldots, N - 1$$

$$T_N = (1-a\Delta t)T_N + a\Delta t\; T_{N-1}$$

$$T_S = T_1$$

In the equations just above, $T_n \equiv T_n(t)$ is the temperature of the front face of the nth slab, at the previous timestep.

TIGN

Eqs. by H.W. Emmons, Oct. 1978.
Modified by H. Mitler, July 1980.
Extended by J. Rockett, Nov. 1980.
In file D; called by NWSTAT.
(Original name of this subroutine was IGNTE1).

Description. When a fuel surface is heated sufficiently (and sufficiently rapidly), the fuel will begin to emit combustible gases (i.e. pyrolyze). If further heated, it will ignite. This sub-subroutine neglects pyrolysis prior to ignition and assumes that the fuel ignites at a specific temperature, T_{ig}. The time at which autoignition occurs (i.e., the time at which this temperature is achieved) is found by linear interpolation (of temperatures) when the surface temperature first exceeds T_{ig}. It then changes the "state of the object" (ISTAT) label for that object.

Piloted ignition: An object will also ignite by contact with a flame. When the horizontal separation between the edge of a flame and a target surface shrinks to zero, the latter might ignite. Since a turbulent flame moves around, we model it as a cylinder, for this purpose. We then must examine their vertical separation. It is assumed that if the target surface lies below the lower surface of the burning fuel slab, it will not ignite; but that it will whenever it is above that (but the bottom of the target slab must not be higher than the flame tip).

Output

ISTAT ; changed from 2 (heating) to 5 (flaming) when T_{ig} is reached.

t_{ig} = TCHNG	Time of ignition.

Required input	From	
$T(t) \equiv$ ZKOZZ(KO)	TMPO	Note: KO = target object; I = flaming object
$T_{ig} \equiv$ ZKOIG(KO)	INPUT	Temperature of (spontaneous) ignition. We have been using 740 °K, but this is subject to change.

$T(t-\Delta t) \equiv$ ZKOZZP(KO)	TMPO	$\theta_2 \equiv$ ZNOZZ(KO)	INPUT
$t \equiv$ ZTZZZ	MAIN	$x_t \equiv$ ZXOZZ(KO)	INPUT
$\Delta t \equiv$ DT	MAIN	$y_t \equiv$ ZYOZZ(KO)	INPUT
$x_o \equiv$ ZXOZZ(I)	INPUT	$h_t \equiv$ ZHOZZ(KO)	INPUT
$y_o \equiv$ ZYOZZ(I)	INPUT	$R_i \equiv$ ZRFZI(KO)	INPUT
$h_o \equiv$ ZHOZZ(I)	INPUT	$R_f \equiv$ ZRFZZ(I)	FIRE
$\theta_1 \equiv$ ZNOZZ(I)	INPUT	$\tan\psi \equiv$ TPSI(I)	FIRE

Calculations	Remarks
If $T(t) > T_{ig}$, set ISTAT(KO) = 5	Spontaneous ignition has taken place.

$$t_{ig} = t - \Delta t \frac{T(t) - T_{ig}}{T(t) - T(t-\Delta t)}$$

Moreover, if $T(t) < T_{ig}$, but

if $R_i + R_f \geq D$ <u>and</u> $-\theta_1 < h_t - h_o < R_f \cot\psi + \theta_2$

where $D = \sqrt{(x_o - x_t)^2 + (y_o - y_t)^2}$,

then set ISTAT(I) = 5 — Piloted ignition has occurred.

and approximate t_{ig} by $t_{ig} \cong t - \frac{\Delta t}{2}$

VENTØ2 and FLOW

Eqs. by H.W. Emmons, October 1978.
Modified by H. Mitler, Dec. 1980.
In file V; called by CALS.

Discussion. This subroutine finds the pressure difference Δp required to satisfy mass conservation in the room. It assumes that there is one room only, though it may (in principle) have any number of vents. (CFC V has up to 5.) The associated sub-subroutine, FLOW, finds the mass flow rates through each vent, given the layer density and height, the vent geometry, and the pressure difference(s) across it. This method of computing a vent flow uses the fact that the two-layer model of room heating always applies pressure drops across a vent which are piece-wise linear from sill to soffit.

The "pieces" are strips whose horizontal boundaries are the clearly recognizable heights such as the sill, the soffit, the layer interface height(s), and the neutral plane(s). For all other physical variables x_i, we can find an explicit expression for x_i, of the form $x_i = f_i(\vec{x})$; i.e., a fixed-point equation. This lends itself to solution by a number of methods. For Δp_f, however, this cannot be done: it is an implicit function of the $\dot{m}_i$'s. Hence in this one case, we must resort to doing some numerical work in a physical subroutine. The flow through each strip is separately calculated and summed appropriately to get the total outflow $\dot{m}_u$ and the total inflow $\dot{m}_d$ (negative) through the vent. The net outflow is $\dot{m} = \dot{m}_u + \dot{m}_d$.

To use this flow formula requires the recognition of the position, properties, and of how many strips exist in the vent. The calculation starts with the pressure drop Δp_f at floor level, across the room wall in which the vent is located. The pressure is specified in units of pressure head in meters of ambient density air above the ambient pressure at the same elevation (above ground level) (not in Newtons/m^2, the S.I. unit!). The distance from the room floor to the bottom of the ith strip is h_i. h_1 is always the height of the sill, h_b. h_2, h_3, etc. give the positions, in succession, of the strips as determined by appropriate heights, such as the soffit, h_t, the cold layer height $\alpha = h_R - h_L$, and the neutral axis position, h_o. We must also know, for each strip, the value of ρ_i, the gas density in the room at the heights $h_i \rightarrow h_{i+1}$. The net mass outflow rate $\dot{m}$ computed above will in general not agree with the <u>correct</u> mass outflow rate $\dot{m}_R$, determined by requiring mass balance in the room:

$$\dot{m}_R = \dot{m}_{out} - \dot{m}_{in} = \dot{E}_L/c_pT_a - \dot{m}_L - \sum_f \dot{m}_f \qquad (1)$$

Agreement is obtained by adjusting Δp_f appropriately. [Eq. (1) was used in subroutine ROOM in CFC III (see p. 89 there) to find $\dot{m}_i$.] Eq. (1) is derived on page 68 of Tech. Report 34. VENTØ2 and FLOW replace VENTØ1, VNT1, VNT2, VNT3, VNT4, ROOM, and REGM, which appear in Mark IV; see remarks at the end of section V.

<u>Output</u>

$\dot{m}_{uv} \equiv$ TMUZZ(KV)	$\dot{m} = \dot{m}_u + \dot{m}_d \equiv$ TMRZZ(KR) = FX
$\dot{m}_{dv} \equiv$ TMDZZ(KV)	$p_f \equiv$ ZPRZZ

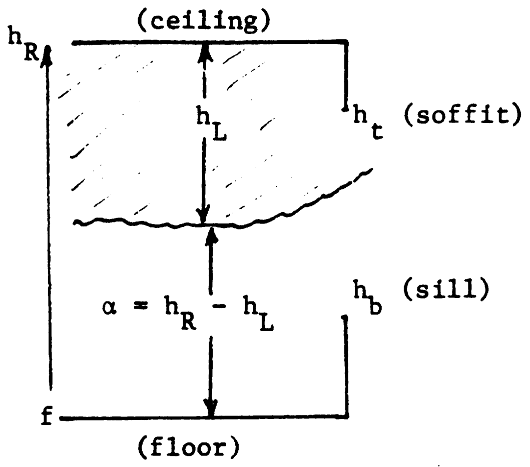
Required input from COMMON	Found or calculated in	Input	From
$h_b \equiv$ ZHBZZ	INPUT	$g \equiv$ G	DATA
$h_t \equiv$ ZHTZZ	INPUT	$p_a \equiv$ ZPAZZ = 0	
$h_L \equiv$ ZHLZZ	INPUT	$\dot{m}_L \equiv$ TMLZZ	LAYR
$h_R \equiv$ ZHRZZ	INPUT	$\dot{E}_L \equiv$ TELZZ	LAYR
$C_d \equiv$ CD	DATA	$c_p \equiv$ ZCAZZ	DATA
B $\equiv$ ZBVZZ	INPUT	$T_a \equiv$ ZKAZZ	DATA
$\rho_a \equiv$ VMAZZ	DATA	$\dot{m}_f \equiv$ TMOZZ(KO)	FIRE
$\rho_L \equiv$ VMLZZ	LAYR		
$p_f \equiv$ ZPRZZ	VENT	Value <u>before</u> iteration step.	

<u>Calculations</u>.

The "cold" layer (in the room) has the same temperature and density as the ambient air (outside the room), i.e., T_a and ρ_a. The hot layer has the density ρ_L. The pressure at floor level at room center is p_f (in meters of ambient air above the ambient pressure p_a at floor level outside); thus, the floor level pressure drop is $\Delta p_f \equiv p_f$ (in meters). That is,

$\Delta p_f = p_{floor} - p_{atmo} = p_f$ Pressure drop from inside to outside across wall (or vent) at floor level (of the room).

Next, we must find the levels h_i in the vent. This is done as follows (refer to the figure at the right): We start at the floor and travel upwards. The pressure drop across the vent remains constant until we hit an interface -- i.e., a density discontinuity. So long as there is no hot layer on the other side of the vent (as is the case here), this will happen at height $\alpha = h_R - h_L$. Now, we define:



$h_1 \equiv h_b$ Bottom of first "piece" (strip) is at vent sill.

If, as in the case shown in the figure, $\alpha > h_b$, then Δp at the sill equals Δp_f, and evidently $h_2 = \alpha$. But if $\alpha < h_b$, then the pressure difference at a height z above the floor is (in these units)

$$\Delta p(z) = p_f + (z - \alpha)\frac{(\rho_a - \rho)}{\rho_a} \qquad \text{for } z \geq \alpha, \qquad (2)$$

and at $h_1 = h_b$,

$$\Delta p_1 \equiv \Delta p(h_1) = p_f + (h_1 - \alpha)\frac{(\rho_a - \rho)}{\rho_a} \qquad \text{for } \alpha < h_1.$$

h_o is the neutral plane height -- from eq. (2),

$$0 = \Delta p(h_o) = p_f + (h_o - \alpha)\left(\frac{\rho_a - \rho}{\rho_a}\right)$$

Then so long as $h_o > h_b$, $h_2 = \min(\alpha, h_o, h_t)$.

If $h_o < h_b$, $h_2 = \min(\alpha, h_t)$.

If $h_2 < h_t$, then there must be an h_3, etc. and we continue to find all the h_i until h_t is reached. Thus denoting the position of the piecewise linear interfaces in the vent as h_i from the sill on up, eq. (2) permits us to write

$$\Delta p_{i+1} = \Delta p_i - \frac{(\rho - \rho_a)}{\rho_a}(h_{i+1} - h_i)$$

Pressure drop at the top of a strip i in terms of that at its bottom; ρ is the interior density.

Note that (2) will become more complicated when (a) the lower layer is allowed to heat up above ambient, and (b) when the vent connects two rooms, with a possible hot layer in the second room.

Now that the strips are determined, the flow through the i^th strip of the v^th vent is given by

$$\dot{m}_i = G_i \frac{|\Delta p_{i+1}|^{3/2} - |\Delta p_i|^{3/2}}{|\Delta p_{i+1}| - |\Delta p_i|} = G_i \frac{|\Delta p_{i+1}| + \sqrt{|\Delta p_i \; \Delta p_{i+1}|} + |\Delta p_i|}{\sqrt{|\Delta p_i|} + \sqrt{|\Delta p_{i+1}|}} \qquad (3)$$

where $\quad G_i = (\text{sgn}\; \Delta p_i)\frac{2}{3}C_d B(h_{i+1} - h_i)\sqrt{2g\rho\rho_a} \quad , \qquad (4)$

$$\rho = \begin{cases} \rho_a & \text{if } \Delta p_i < 0 \\ \rho_i & \text{if } \Delta p_i \geq 0 \end{cases} \quad \text{and} \quad \rho_i = \begin{cases} \rho & \text{above the interface} \\ \rho_a & \text{below the interface} \end{cases}$$

The flows are calculated in this way for each vent in the room, and then summed:

$\dot{m}_u = \Sigma' \, \dot{m}_i = \sum_v \dot{m}_{uv}$ Outflow from hot (upper) layer

$\dot{m}_d = \Sigma'' \, \dot{m}_i = \sum_v \dot{m}_{dv}$ Flow of lower layer (d stands for "down").

and $\dot{m} = \dot{m}_u + \dot{m}_d = \Sigma \, \dot{m}_i$

The sums are over all vents and all strips in each vent; a single prime indicates that hot (out-)flows only are to be included; a double prime indicates that we must include only cold flows.

Note: $\dot{m}_j > 0$ means outflow, $\dot{m}_j < 0$ means inflow; j = i or d or u.

The above level classification and flow calculations are made for Δp_f and $\Delta p_f + \varepsilon$, for which the net flows are $\dot{m}(p_f)$ and $\dot{m}(p_f+\varepsilon)$, respectively. Then the value of Δp_f is linearly adjusted to produce $\dot{m}_R$, as given by eq (1). The obvious way to do this is to set

$$(\Delta p_f)_{adj} = \Delta p_f + \frac{\dot{m}_R - \dot{m}(p_f)}{\dot{m}(p_f+\varepsilon) - \dot{m}(p_f)}\varepsilon \qquad (5)$$

This is precisely what the Newton-Raphson method gives, if we define

$$f(p_f) \equiv \dot{m}_R - \dot{m}(p_f) \quad ,$$

and ask for a zero of f(x). However, because $\dot{m} \propto \sqrt{p}$ (rather than being linear or of a higher power), the correction term thus added to the initial guess Δp_f is twice as large as it ought to be, and thus use of eq. (5) frequently leads to instability. A better expression to use is thus

$$\bar{p}_f \equiv (\Delta p_f)_{adj} = \Delta p_f + \frac{\varepsilon}{2} \frac{\dot{m}_R - \dot{m}(p_f)}{\dot{m}(p_f+\varepsilon) - \dot{m}(p_f)} \tag{6}$$

The corresponding flows will (to first order) be given by

$$\left.\begin{aligned} \bar{\dot{m}}_{uv} &= \dot{m}_{uv} + \left[\frac{\dot{m}'_{uv} - \dot{m}_{uv}}{\varepsilon}\right](\bar{p}_f - p_f) \\ \text{and} \quad \bar{\dot{m}}_{dv} &= \dot{m}_{dv} + \left[\frac{\dot{m}'_{dv} - \dot{m}_{dv}}{\varepsilon}\right](\bar{p}_f - p_f) \end{aligned}\right\} \tag{7}$$

$\dot{m}_{uv}$ is the upper flow found in the vth vent with $\Delta p = p_f$, and $\dot{m}'_{uv}$ is that found with $\Delta p = p_f + \varepsilon$. Similarly for $\dot{m}_{dv}$.

So long as the layer thickness is smaller than any of the room vent transoms -- i.e.,

$$h_L < \min_{v} \{h_t(v)\} , \tag{8}$$

then we must have

$$\left.\begin{aligned} \dot{m}_u &= 0 \\ \text{and} \quad \dot{m}_d &= \dot{m}_R \quad , \end{aligned}\right\} \tag{9}$$

immediately. It also follows that

$$p_f = (\operatorname{sgn} \dot{E}_L)\left(\frac{2\dot{m}_R}{3G_1}\right)^2 \tag{10}$$

(although this is of small importance, since we only need the pressure difference <u>in order</u> to obtain $\dot{m}_d$). The use of eqs. (9) and (10) eliminates the need to use eqs. (3), (6), and (7) -- i.e., to call FLOW twice, etc. Moreover, it will not be necessary to iterate repeatedly in order to have p_f (generally the most sensitive variable) converge -- it will only be necessary that $\dot{m}_R$ (and therefore $\dot{E}_L$, $\dot{m}_L$, and $\dot{m}_f$) converge to the appropriate accuracy.

LAYRØ4

Equations by H. Mitler, June 1979; modified, Dec. 1980

In file L; called by CALS

Description

Aside from the $CaCO_3$ filler, the chemical composition of flexible polyurethane foam #7004 is $C_{3.80}H_{7.26}N_{0.21}O$. A. Tewarson has measured the mass fractions of smoke, CO, CO_2, and unburned hydrocarbons emanating from the burning foam (exclusive of the inert $CaCO_3$ filler). These are, respectively, f'_s, f'_{CO}, f'_{CO_2}, and f'_{HC}. The smoke is a combination of soot and condensed hydrocarbons. For our purposes we can add the (small) gaseous hydrocarbon fraction f'_{HC} to f'_s as well. We assume no ash residue (other than $CaCO_3$) upon combustion. The version of LAYR in Mark 4 (LAYRØ3) is identical to the LAYRØ2 subroutine used in CFC III, but it also calculates the CO, CO_2, and smoke concentrations in the layer, using these data. Since the data are given *via* INPUT, the calculation applies generally, rather than just to PU 7004 (though that is the default). The present version has the following further changes: First, we use mass fractions relative to the (combustible part of the) foam, rather than to the carbon mass. Second, we can infer the approximate mass of water vapor produced in the (incomplete) combustion, and have therefore calculated the H_2O content in the layer. Finally, of course, we have included LIMITS here. The derivation of the equations for oxygen concentration is given on pp. 19-23 of T.R.34. The concentration of smoke calculated here is about 15% higher than it is in reality, because the deposition of soot on ceiling and walls has been neglected here; see comments under ABSRB3.

Output

$m_L \equiv$ ZMLZZ	$\dot{E}_L \equiv$ TELZZ	$Y_O \equiv$ ZYLOZ	$Y_{CO_2} \equiv$ ZYLDZ
$\dot{m}_L \equiv$ TMLZZ	$h_L \equiv$ ZHLZZ	$Y_W \equiv$ ZYLWZ	$Y_s \equiv$ ZYLSZ
$E_L \equiv$ ZELZZ	$T_L \equiv$ ZKLZZ	$Y_{CO} \equiv$ ZYLMZ	

Input Required from Common	Originating in Subroutine:
$\dot{m}_p \equiv$ TMPZZ(KO)	PLUM
$\dot{E}_p \equiv$ TEPZZ(KO)	PLUM
$\dot{E}_e \equiv$ TEUZZ(KV)	VENT
$\dot{E}_c \equiv$ TELZD(KR)	CNVL
$\dot{E}_R \equiv$ TELZR(KR)	RDNL
$\dot{m}_e \equiv$ TMUZZ(KV)	VENT

Input Required from Common	Originating in Subroutine:
$\dot{m}'_a \equiv$ TMPLU(KO) = 0	PLUM
$-\dot{m}_f \equiv$ -TMOZZ(KO)	FIRE
$C \equiv$ ZCAZZ	INPUT
$L = L_x \equiv$ ZLRZX	INPUT
$W = L_y \equiv$ ZLRZY	INPUT
$T_a \equiv$ ZKAZZ	INPUT
$\rho_a \equiv$ VMAZZ	INPUT
$\chi \equiv$ CHI	INPUT

$\gamma_s \equiv$ XGAMAS	INPUT	For the P.U. foam, $\gamma_s = 9.85$
$\gamma \equiv$ XGAMMA	INPUT	For this foam, $\gamma = 14.45$
$f'_{CO} \equiv$ FCO	INPUT	Tewarsen's results for P.V. foam 7004 implies $f'_{CO} = .0133$
$f'_{CO_2} \equiv$ FCO2	INPUT	For 7004, $f'_{CO_2} = 1.504$
$f'_W \equiv$ FH20	INPUT	For 7004, $f'_W = 0.7135$
$f'_s \equiv$ FS	INPUT	For PU # 7004, $f'_s = 0.241$, where we have included condensed and uncondensed hydrocarbons as well as soot, for Mark 5.

Calculations

$\dot{E}_L = \dot{E}_p - \dot{E}_e + \dot{E}_c + \dot{E}_R$	Rate of change of energy of hot layer
$\dot{m}_L = \dot{m}_p - \dot{m}_e$	Rate of change of mass of hot layer
$E_L = \int_0^t \dot{E}_L \, dt$	Energy of hot layer
$m_L = \int_0^t \dot{m}_L \, dt$	Mass of hot layer
$h_L = \dfrac{E_L}{LWCT_a\rho_a}$	Depth of hot layer
$T_L = \dfrac{E_L}{m_L C}$	Temperature of hot layer

$\dot{m}_t \equiv \dot{m}_{test} = -[(\gamma\chi + 1)\dot{m}_f + \dot{m}_p]0.2318/Y_0$ See eq. (25), p. 22 of T.R.34

$$\dot{m}_{ox} = \begin{cases} 0.2318[\dot{m}_p + \dot{m}_f(\gamma_s\chi + 1)] - Y_o\dot{m}_e \quad , & \dot{m}'_a \geq \dot{m}_t \\ 0.2318(\dot{m}_p + \dot{m}_f)\left(1 - \frac{\gamma_s}{\gamma}\right) - Y_0\left[\dot{m}'_a \frac{\gamma_s}{\gamma} + \dot{m}_e\right] \quad , & \dot{m}'_a < \dot{m}_t \end{cases} .$$

The first equation (just above) applies when enough oxygen is entrained (including, possibly, a contribution from the part of the plume in the hot layer) to allow combustion up to the open-air limit χ. This is eq. (22), found on page 19 of T.R.34. When $\dot{m}'_a = 0$, the criterion $\dot{m}'_a < m_{test}$ (see second equation) becomes $0 < \dot{m}_{test}$. But that is just the criterion for oxygen starvation.

When $\dot{m}'_a > 0$, the second equation corresponds to the case where oxygen starvation occurs even when (non-vanishing) oxygen entrained into the plume from the *layer* is included. This equation, incidentally, is eq. (27), on p. 22 of T.R.34.

NOTE: These three equations are unnecessarily complicated, and will be fixed in later versions!

$m_{ox} = \int_o^t \dot{m}_{ox}(t')\, dt'$	Mass of oxygen in the layer
$Y_o = m_{ox}/m_L$	Mass fraction of oxygen in the layer
$\dot{m}_{CO} = -\dot{m}_f f'_{CO} - \dot{m}_e Y_{CO}$	Rate of change of CO mass in the layer
$\dot{m}_{CO_2} = 0.0005(\dot{m}_p + \dot{m}_f) - \dot{m}_f f'_{CO_2} - \dot{m}_e Y_{CO_2}$	Rate of change of CO_2 mass in the layer
$\dot{m}_s = -\dot{m}_f f'_s - \dot{m}_e Y_s$	Rate of change of smoke mass in the layer
$m_{CO_2}(t) = \int_0^t \dot{m}_{CO_2}(t')\, dt'$	Mass of CO_2 in the layer
$Y_{CO_2} \equiv \text{ZYLDZ} = m_{CO_2}(t)/m_L(t)$	Mass fraction of CO_2 in the layer (initial value = 0.0005)
$m_{CO}(t) = \int_0^t \dot{m}_{CO}(t')\, dt'$	Mass of CO in the layer
$Y_{CO} \equiv \text{ZYLMZ} = m_{CO}(t)/M_L(t)$	Mass fraction of CO in the layer (initial value = 0.0)
$m_s(t) = \int_0^t \dot{m}_s(t')\, dt'$	Mass of smoke in the layer

$Y_s(t) \equiv \text{ZYLSZ} = m_s(t)/m_L(t)$	Mass fraction of smoke in the layer (initial value = 0.0)
$\dot{m}_W = -\dot{m}_f f'_W - \dot{m}_e Y_W$	Rate of change of water mass in the layer
$m_W(t) = \int_0^t \dot{m}_W(t')\, dt'$	Mass of H_2O (gas) in the layer
$Y_W t) = m_W(t)/m_L(t)$	Mass fraction of H_2O in the layer (initial value = 0.0)

ABSRB1

Equations by H. Mitler, July 1979

In file L; called by CALS

Description

This subroutine calculates the infrared absorptivity of the hot layer. In this version (used in CFC III), we approximated it by a simple exponential fit to the data on the smoke attenuation coefficient, obtained from the third full-scale bedroom fire test (1975), channel 153. In CFC III, however, we used its reciprocal, the absorption length λ.

Output	Input Needed from Common	From Subroutine
$\kappa \equiv$ ZULZZ(KR)		
	$t \equiv$ ZTZZZ	MAIN

Calculation

$$\kappa = 0.001\, e^{0.0286t}$$

ABSRB2

Equations by H.E. Mitler, July 1979

In file L; called by CALS

Description

This subroutine calculates the infrared absorptivity of the hot layer. We assume the absorption coefficient is in fact grey and due mostly to the smoke concentration in the layer. The proportionality constant used in the equation below was obtained by comparing the IR absorptivity as measured in the July 6, 1977 full-scale fire test (channel 126), with the smoke concentration calculated by LAYRØ4 in a standard run. (Note, however, that that concentration is too high--see comments in LAYRØ4 and ABSRB3).

Output	Input Needed from Common	From Subroutine
$\kappa \equiv$ ZYKZZ(KR)	$Y_s \equiv$ ZYLSZ(KR)	LAYR

Calculation

$\kappa = 265 Y_s$

ABSRB3

Equations by H.E. Mitler, Dec. 1980

In file L; called by CALS

Description

The absorptivity of the hot layer is here found from first principles (albeit with simplifications). The absorption coefficient of soot, k_o, is found from its value at 0.94 microns. The absorptivity of H_2O and CO_2 is found in a wide-band approximation. The soot concentration calculated by Mark V is a bit too large. This is so because about 14% of the soot is deposited on ceiling and walls, but that has not been taken into account in y_s. For purposes of this calculation, therefore, the calculated y_s

(mass fraction of soot) is multiplied by 0.857 (the soot mass equation in LAYR will be modified correctly in version VI).

ABSRB3 calls subroutine EMSVTY and its associated sub-subroutine, which do most of the work.

Output

$\kappa = \kappa_s + \kappa_g$ = ZULZZ(KR)

Input Needed from Common	From Subroutine	Description
$y_i \equiv$ ZYLiZ	LAYR	Mass concentration of species i in the layer. i = O,D,M,S (for oxygen, CO_2, CO, and smoke, respectively).
$T_L \equiv$ ZKLZZ(KR)	LAYR	Layer temperature (in °K)

Calculations	Remarks
$k_o = 1.3147\ (.857)\ y_s/T_L$ $= 1.1266 \times 10^6\ y_s/T_L$	This gives the absorptivity of the soot (to 0.94 μ radiation).
$\sigma = \sum_i'' \frac{y_i}{m_i} = \frac{y(O_2)}{32.000} + \frac{y(CO_2)}{44.011} + \frac{y(CO)}{28.011} + \frac{y(H_2O)}{18.016}$	
$\sigma' = \sum_i'' y_i = y(O_2) + y(CO_2) + y(CO) + y(H_2O)$	
$\Sigma = .035555 + \sigma - \sigma'/28.016$	
$PCO2 = \frac{y(CO_2)}{44.011\Sigma}$	Partial pressure of CO_2 in the layer, in units of atmospheres.
$PH2O = \frac{y(H_2O)}{18.016\Sigma}$	Partial pressure of H_2O in the layer.

Then we CALL EMSVTY (k_o, 1, T_L, PCO2, PH20, EMISS)

This returns the emittance of a unit-path-length layer,

ε = EMISS. Then

$\kappa = -\ln(1-\varepsilon)$. Total absorption coefficient of layer.

EMSVTY

(also SOOT, DLECK, PENTA, CHEBY, EGAS, and SCRTCH).
Eqs. by Ashok T. Modak, July 1978.
Called by ABSRB3.

Description. This subroutine computes the emissivities of isothermal and homogeneous mixtures of soot, CO_2, and H_2O at a total pressure of 1 atmosphere. The accuracy of the calculation is better than 5% as compared with spectral calculations and experimental measurements. The limits of validity are 300 to 2000 ^oK, a gas partical pressure range 0 to 1 atmosphere, and a pressure-pathlength of 0 to 5.98 atm-meters. Each call on EMSVTY takes 11.85 ms. of CPU time on an IBM 370/158.

Output	Description
$\varepsilon \equiv$ EMISS	Dimensionless emissivity (emittance) of the mixture.

Input needed (from COMMON)	Source	Remarks
$k_o \equiv$ ZKLED	ABSRB3	The absorption coefficient of soot at a wavelength of 0.94 μ (in m^{-1}).
$L \equiv$ PATHL	ABSRB3	Mixture pathlength (in m). Must be ≥ 0.
$T \equiv$ ZKLZZ	LAYR	Gas temperature (degrees Kelvin). We must have $300 \leq T \leq 2000$.
PCO2	ABSRB3	Partial pressure of CO_2 (in atmospheres) in a gas mixture at a total pressure of 1 atm. Must have $0 \leq PCO2 \leq 1.0$. For $PCO2 < .0011$, $PCO2.L < .0011$, and $PCO2.L > 5.98$, the contribution of CO_2 to the mixture emissivity is set equal to zero.
PH2O	ABSRB3	Partial pressure of water vapor. Same remarks apply as for PCO2. Moreover, we must have $PCO2+PH2O \leq 1.0$.

Calculations

It operates by making a number of calls to subroutines and functions.

EMSVTY is called with the arguments (k_o,L,T,PCO2,PH2O,EMISS). It immediately calls sub-subroutine

SOOT(k_o,L,T,τ_s), which returns τ_s, the soot transmissivity.

Subroutine SOOT utilizes sub-subroutine PENTA, which computes the pentagamma function (because the spectrally integrated emissivity of a pathlength L of soot is given by

$$\varepsilon_s = 1 - \frac{15}{\pi^4} \psi^{(3)} \left(1 + \frac{\lambda_o k_o TL}{C_2}\right)$$

where $\psi^{(3)}$ is the pentagamma function; λ_o = 0.94μm, and C_2 is the second Planck constant. See reference (31)). PENTA uses ASYMP, a sub-subroutine which computes the asymptotic expansion for $\psi^{(3)}$.

In obtaining the argument for the pentagamma function, Modak uses the (approximate) equation

$$\lambda_o k_o = 7f_v,$$

obtained from numerous correlation studies. I then note that f_v is trivially obtained from

$$f_v = Y_s^* \rho_L/\rho_s$$

where ρ_L is the layer (gas) density, y_s^* the (correct) soot mass fraction, and ρ_s the mean soot density. For Y_s^* I use 0.857 y_s (see discussion for ABSRB3), while for ρ_s the estimate 2.0 g/cm^3=2000 kg/m^3 is reasonable. Hence the expression for k_o used in the calculation section of subroutine ABSRB3.

Having τ_s, we immediately get

$$\varepsilon_s = 1 - \tau_s.$$

Next, ε_g is calculated by FUNCTION EGAS, which computes the emissivity (emittance) of a given pathlength L of a mixture of CO_2 and H_2O at temperature T:

$$\varepsilon_g = \text{EGAS(L,PCO2,PH2O,T)}$$

This function itself calls sub-subroutine SCRTCH, which gives the separate contributions of CO_2 and H_2O to ε_g, and the FUNCTION DLECK, which computes the 2.7 and 15μ overlap correction for mixtures of CO_2 and H_2O. This is computed by using a temperature-adjusted version of the overlap correction factor suggested by B. Leckner[30].

Finally, subroutine SCRTCH also calls sub-subroutine CHEBY, which gives values of various Chebyshev polynomials (used in curve-fitting, above).

The reader interested in the extensive details of this calculation should look in references (22), (30), (31), and the listings of these subroutines, in Appendix D.

PLUM

Eqs. by H.W. Emmons, June 1977.
Modified by H.E. Mitler, July 1978.

Description. This subroutine calculates the dimensions of the plume above the fire and the mass and energy transported by the plume into the hot layer. The density and velocity profiles are Gaussian, and the equations essentially those found by Morton, Taylor and Turner (ref. 20), for a point source of heat. This formulation assumes an area source of heat of given radius by taking a virtual point source at an appropriate distance below the fire base, and subtracting from the resulting mass flux the amount of air which would have been entrained between the vertex and the fire base. The plume is assumed to stop at the layer interface. The entrainment coefficient α is assumed to be constant. The equations are discussed more fully in Tech. Report 34, pp. 10-13. Except for minor programming modifications, this is still the same as PLUMØ2 in CFC III.

Output for each plume (KO):

h_p ≡ ZHPZZ(KO)	$\dot{E}_p$ ≡ TEPZZ(KO)
$\dot{m}_p$ ≡ TMPZZ(KO)	$\dot{m}'_e$ ≡ TMPLU = 0 in this subroutine

Input required from COMMON	Found in subroutine	Input	From
h_R ≡ ZHRZZ(KR)	INPUT	c_p ≡ ZCAZZ	DATA BLOCK

Input (cont'd)	From	Input	From
$h_F \equiv$ ZHOZZ(KO)	INPUT	$T_a \equiv$ ZKAZZ	DATA
$h_L \equiv$ ZHLZZ(KR)	LAYR	$\rho_a \equiv$ VMAZZ	DATA
$R \equiv$ ZRFZZ(KO)	FIRE	$\dot{m}_f \equiv$ TMOZZ(KO)	FIRE
$g \equiv$ G	DATA	$\dot{E}_{PR} \equiv$ TEPZR(KO)	RDNP
$\dot{E}_f \equiv$ TEOZZ(KO)	FIRE	$\alpha \equiv$ ALFA (=0.1)	DATA

Calculations	Remarks
$h_p = h_R - h_F - h_L$	Height of plume
$x = R/1.2\alpha$	Distance of virtual point source below base of fire
$H_p = h_p + x$	Height of augmented plume -- i.e., distance between virtual point source and layer
$b = 1.2\alpha H_p$	Plume radius at layer
$C_o = 25g\dot{E}_f/48\pi\alpha^2 c_p T_o \rho_o$	A convenient coefficient
$u = (C_o/H_p)^{1/3}$	Axial velocity of plume gases at top of plume
$u_f = (C_o/x)^{1/3}$	Axial velocity of plume gases at fire base
$\dot{m}_p = \pi\rho_o(b^2u - R^2u_f) - \dot{m}_f$	Net mass flow into layer (augmented by pyrolysis fuel mass)
$\dot{E}_p = \dot{m}_p c_p T_o - \dot{E}_f - \dot{E}_{PR}$	Convective energy flow rate into layer due to mass transfer from the plume

Subroutines ROOM, VENTØ1, REGM, and VNT4 are only valid for Mark 4, and do not exist in Mark 5 any longer. Any user in possession of a Mark 4 tape who wants the documentation for these (former) subroutines, please contact the author.

VI. PRESENT LIMITATIONS OF MARK 5

In this section I will discuss some of the parameter limits of the program, as well as its limitations. Like the program itself, its limitations fall into three categories: mathematics, physics, and structure. From the point of view of mathematics, it has the same difficulties every other program designed to solve a simultaneous set of nonlinear equations has -- that is, there is a small percentage of cases where it fails to converge, at some point in the run. A more serious difficulty is where the "convergence" is spurious; this is more serious because we cannot always tell that the answers are incorrect. (On the other hand, the close coupling of the equations sometimes will lead to a "healing" of incorrect answers! That is, the errors do not always propagate.)

Both of these difficulties are illustrated by the following set of runs: for a run with the standard fuel (and target) in a huge room -- of dimensions 400 x 600 x 400 m -- the layer temperature appears to take a sudden jump at t = 80, from 300.00 to 307.04 ^{o}K, and the pressure at the floor jumps simultaneously to the clearly spurious value Δp = 100 meters (the limiting value it can take on). For a smaller -- but still very large -- room, of dimensions 200 x 300 x 200 m, T_L suddenly jumps to 305.44 [and Δp to 100] at t = 200. For the case where the room dimensions are (L_x, L_y, L_z) = (100, 150, 100) m, the calculation is quite reasonable until t = 320, at which point Δp goes to 100. Examining the limits of validity from the other direction, we find that for rooms of dimensions (100 x 150 x 50), (100 x 150 x 75), and (100 x 150 x 85), there are no difficulties, out to 500 sec. For 100 x 150 x 95, however, we do not get a spurious result, but fail to converge at 348.7 sec, though all the answers up to that point appear to be reasonable. A systematic investigation of the results for rooms of different dimensions reveals no simple rule by which we can determine beforehand which calculation will go through and which not, unfortunately.

Next: a gas burner with R = 0.8 m and $\dot{m}_{gas}$ = 10 gm/sec runs through (fairly reasonably) out to 500 sec. Yet for R = 1.0 m, the calculation fails to converge at t = 21.56 sec, and we are forced to terminate the run.

Again, a pool fire (with P.U. foam) of radius R = 0.8 m in the standard room runs quite reasonably out to 180 sec. However, it then runs into trouble beyond 181.5 sec -- it will not converge.

Each of these cases, as we can see, is fairly extreme -- yet for the last case, the difficulty only arose well into the fire, when everything should have been settling down nicely.

The nature of the difficulties is varied. One possible source is our convergence criterion. It is a reasonable one, but certainly does not guarantee that we have indeed converged to the solution of our equations. This occasionally leads to apparent "convergence" to incorrect values, and then extrapolation to a new timestep will lead to a very poor set of approximations, possibly leading to failure to converge further. At other times, subroutines may have been written in such a fashion that the program will loop indefinitely, at some point. Finally, we may simply be in a region where the Jacobian is either singular or very ill-conditioned. Evidently each of these requires a different method of resolution, but they all lie outside the scope of this document. Of course we cannot rely on the person who writes a subroutine to be able to anticipate every possible numerical difficulty which his subprogram could lead to. We must therefore rely on very robust numerical algorithms. Some tricks which will sometimes sidestep a numerical difficulty are discussed in section VIII, Use of the Program.

One structural difficulty, of a fairly trivial nature, is that the INPUT routine is not yet sophisticated enough to realize when inconsistent input is being fed to the program. Thus, it is easy to inadvertently put an object outside of the room (which will lead to a "lethal" arithmetic error), for example.

Another structural limitation is the total number of vents and objects which can be input. In order to put in more, the program would have to be recompiled with larger DIMENSION statements.

Finally, the program is not as flexible as it might be -- for example, it does not permit input of ε = TOLER, should we want to change it from its present value of 3×10^{-4}. Nor does it permit switching to a Jacobi algorithm. The initial apex angle of the cone is still fixed at 30°. The program is not as modular as we would like, either. Nor can we make a series of runs where only one parameter takes on several values, without having to reinput everything.

The limitations of the physics are the most obvious, perhaps. Among these are: no provision for vents in floor or ceiling; no provision for heating the lower gas layer (mostly by mixing with the hot layer at the vent, presumably); the limitation to a single room; no criterion for flashing of the upper layer. The convective heat transfer coefficients are too crude. These and many more are of course being planned for Mark 6 and later versions.

VII. COMPARISON BETWEEN THEORY AND EXPERIMENT.

In order to validate a theory or a model, we must of course be able to predict, within specified limits of accuracy, the results of an experiment. A great many full-scale fire tests have been run, but only a small fraction of those have been well instrumented. In 1977, Factory Mutual Research Corporation ran a series of seven thoroughly instrumented fires(32) whose results could then be compared with our calculations. A comparison of two of the tests with predictions made by CFC III was reported by Emmons(4). A comparison between calculated and experimental results, using Mark 5, for tests 0, 1, 2, and 6 is given in Mitler(33) and Mitler & Rockett(38). A very abbreviated version of those two papers will be given in this section. The "standard configuration" was a room 8' x 12' x 8', with a 30" x 80" doorway as vent, which contained two objects. Object number 1 was a 5' x 5' x 4" slab of polyurethane foam (#7004) in one corner of the room; its top surface was 2 ft. above the floor. Object number 2 was a 4' x 1' x 4" slab of the same material, 10" higher and in a facing corner (see fig. 35). Slab #1 was ignited at its center, and the progress of the fire observed. The appearance of the room during the fire, and some of the variables involved, are shown in fig. 36. This case was run (at Factory Mutual's test facilities) three times, and the tests were labeled # 0, 1, and 2. Test #6 was identical, but had a window replacing the doorway.

First, it must be noted that the experimental results fluctuate substantially. Thus, tests 0, 1, and 2 were meant to be identical. Yet the layer height recorded in the three cases varied substantially -- see fig. 37.

The pyrolysis rate had similar variations -- indeed, the fluctuations

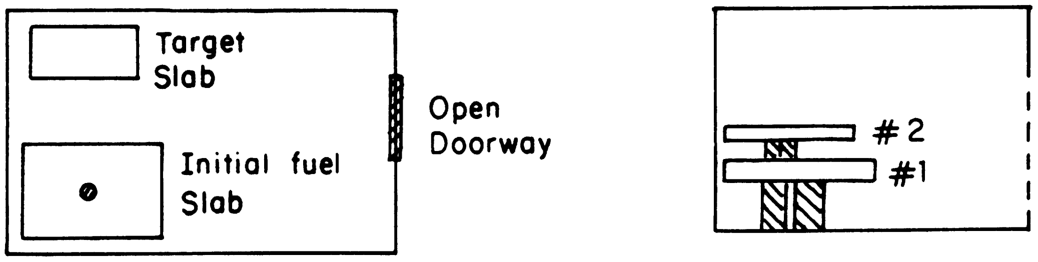


Fig. 35. Schematics of the standard configuration for fire tests 0, 1, 2. Top and side views, respectively.

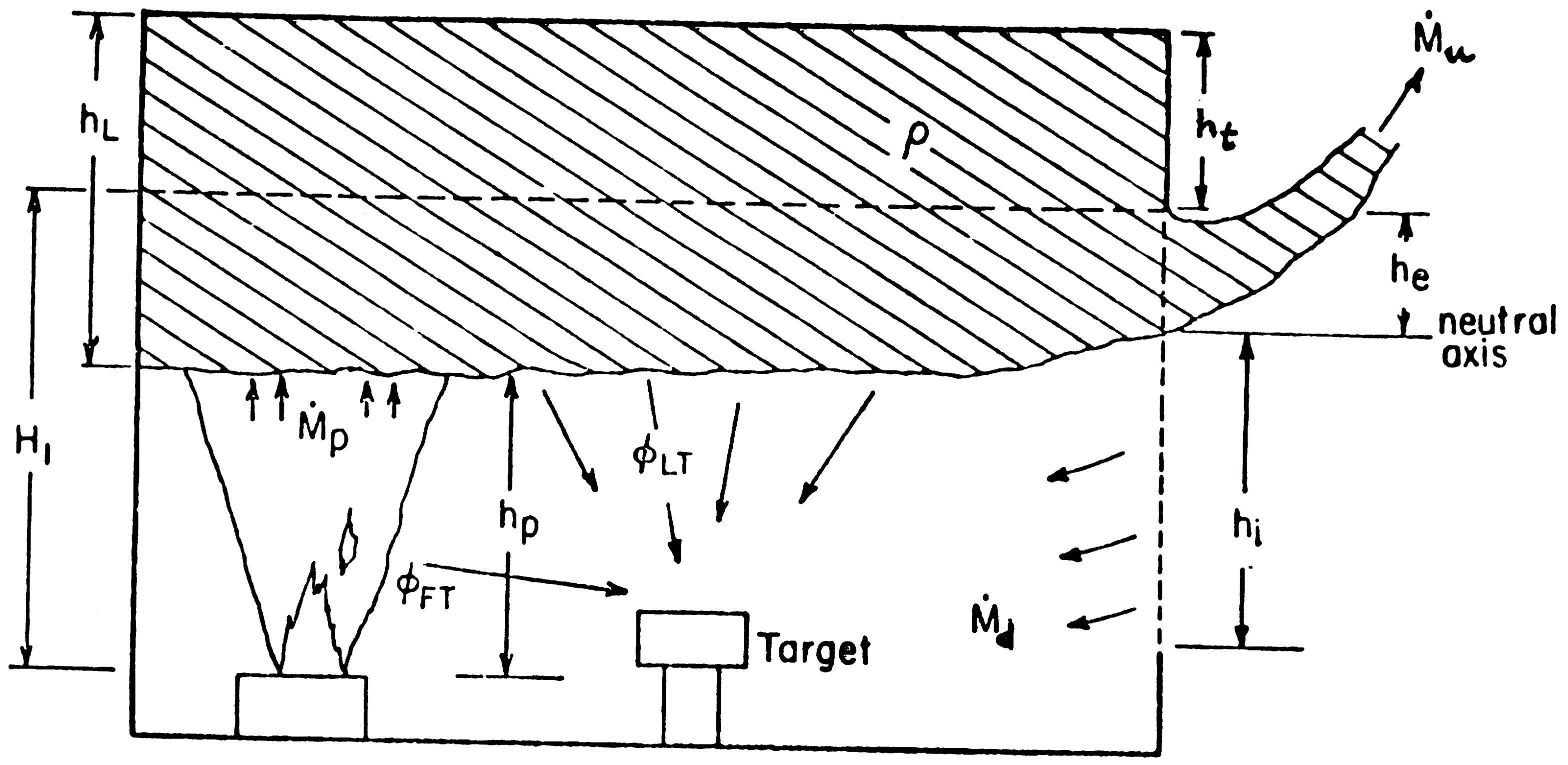


Fig. 36. Schematic of the enclosure, showing mass fluxes in and out of the room, mass flux from plume into layer, and radiative fluxes to the target from layer and flame.

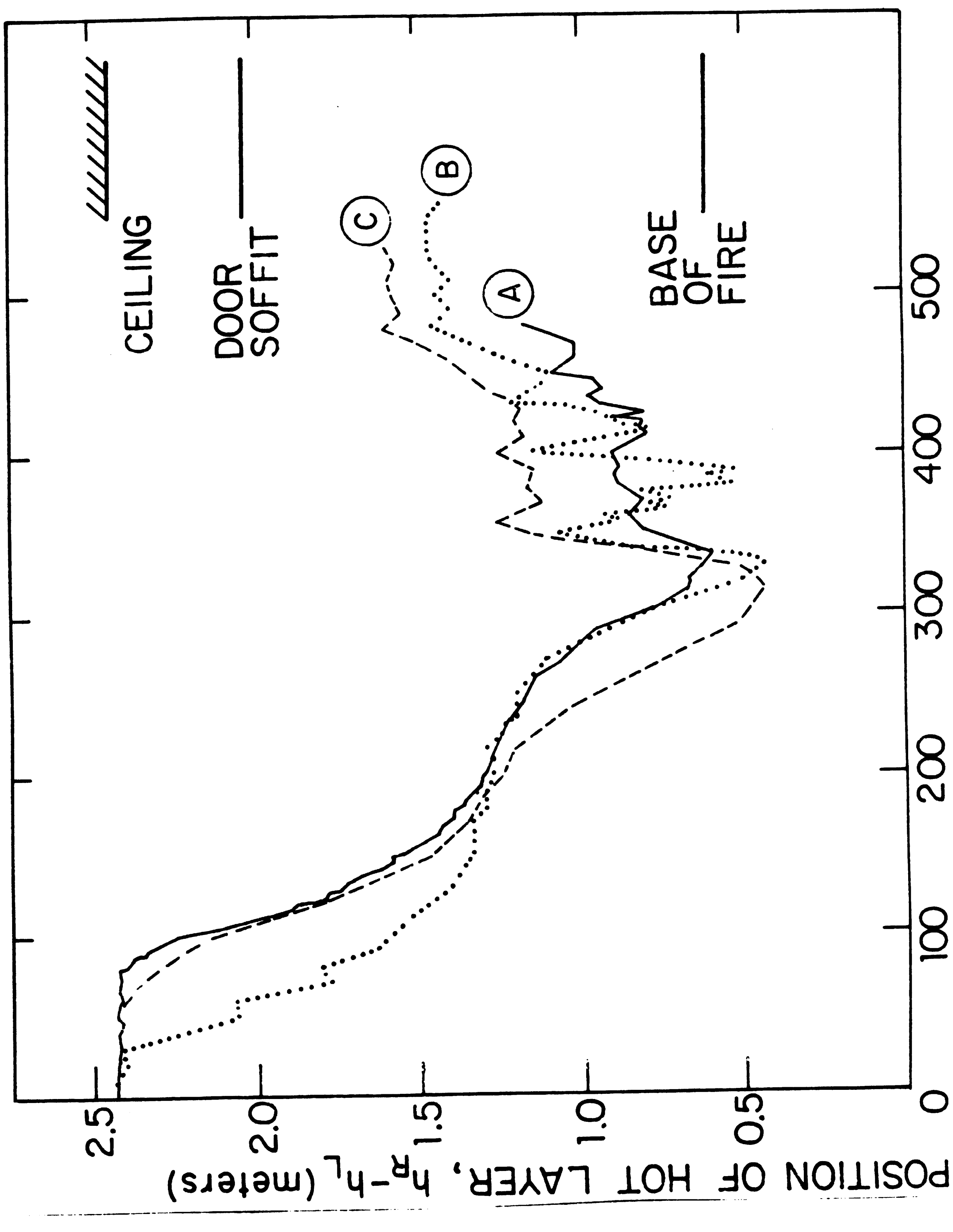


Fig. 37. Depth of hot layer in the room, for tests #0, 1, and 2 (labeled A, B, and C, respectively).

for each of the three runs were very large for $t < 100$ to 150 sec. With such differences among supposedly identical runs, it is clear that we cannot have any better agreement between a theoretical calculation and any single run.

Second, the way the raw data is handled also affects how well "experiment" and "prediction" agree. Thus, "the" mean hot layer temperature was calculated in different ways by us and by the Factory Mutual group: they took the mean hot layer temperature to be the average temperature (as measured by six thermocouples) at a given height -- 30% of the room height down from the ceiling. That gives the solid curve in fig. 38. We calculated the mean layer temperature and depth simultaneously, from the (vertical) temperature profiles as given by three racks of thermocouples in the room. That result is given by the dashed curve in fig. 38 and is seen to deviate considerably from the first curve; at their peaks, the temperatures differ by 150^{o} C -- this will lead, for example, to a factor of more than two difference in the radiative emission from the layer, and a comparable factor between prediction and experiment, therefore.

With these important caveats, we compare theory and experiment for the standard run, in figs. 39-42. We see that the agreement is generally good, except for the carbon monoxide concentration, as seen in fig. 41. The smoke concentration was not measured directly, but the resulting emissivity of the layer seems to be good, as the measured and predicted fluxes due to the ceiling layer agree very well (fig. 42).

The predictions for test #6, where a window replaces the doorway, are not quite so good, but are still acceptable.

Perhaps the poorest prediction we make is of the burnout of the fuel, which was made ad hoc; this is not of great significance now, but clearly has to be improved. The most puzzling question is why the layer temperature is well predicted for most of the burn, but underpredicted later on. There are several possible reasons, each of which involves some physics not yet incorporated into the program (such as burning in the layer). Finally, the CO concentration is badly predicted. This reflects our ignorance of the relevant mechanisms, at present.

Aside from these points, the agreement is very good, and suggests that much of the physics has been put into the program fairly well.

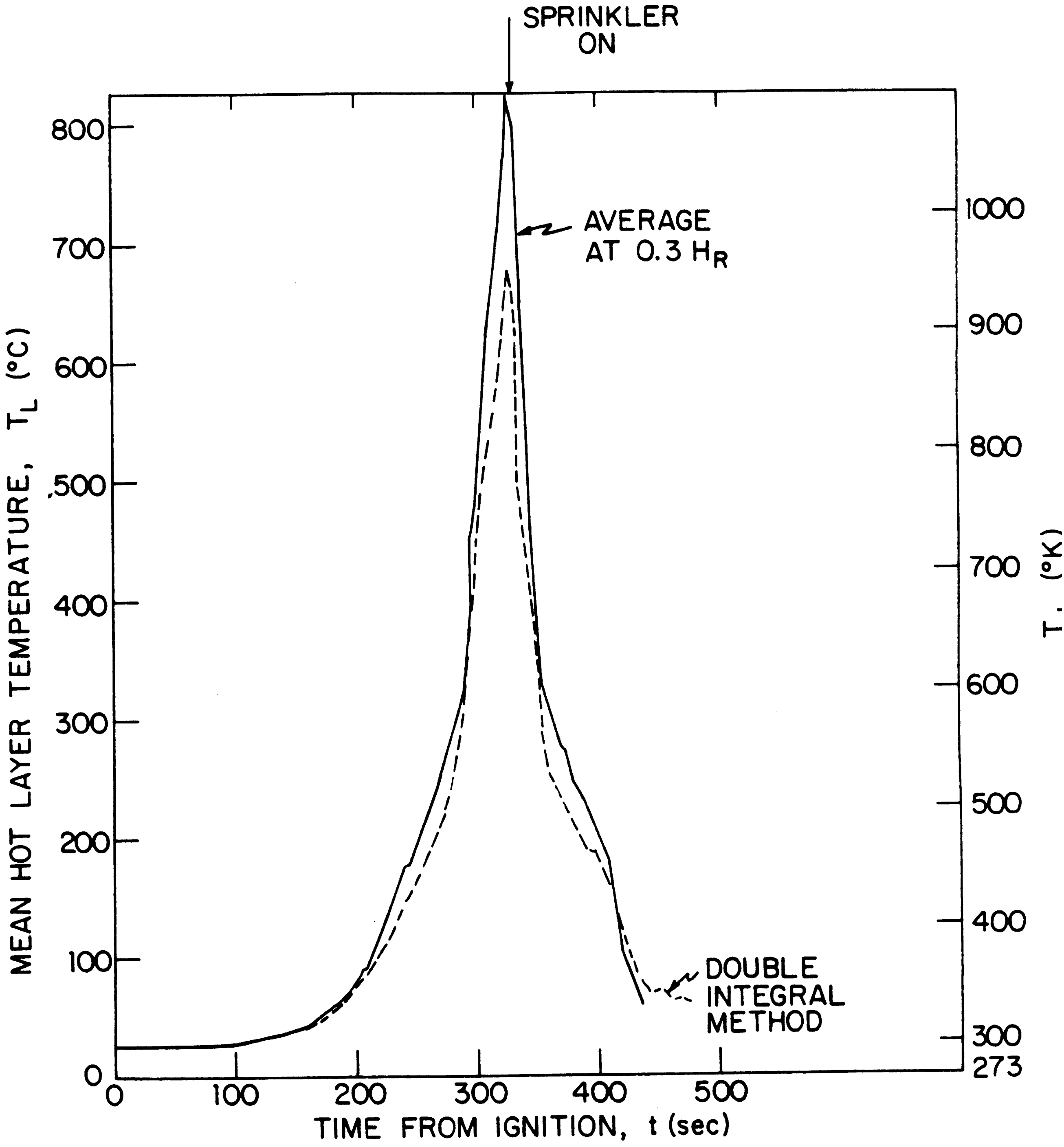


Fig. 38. The hot layer temperature T_L, measured in two ways.

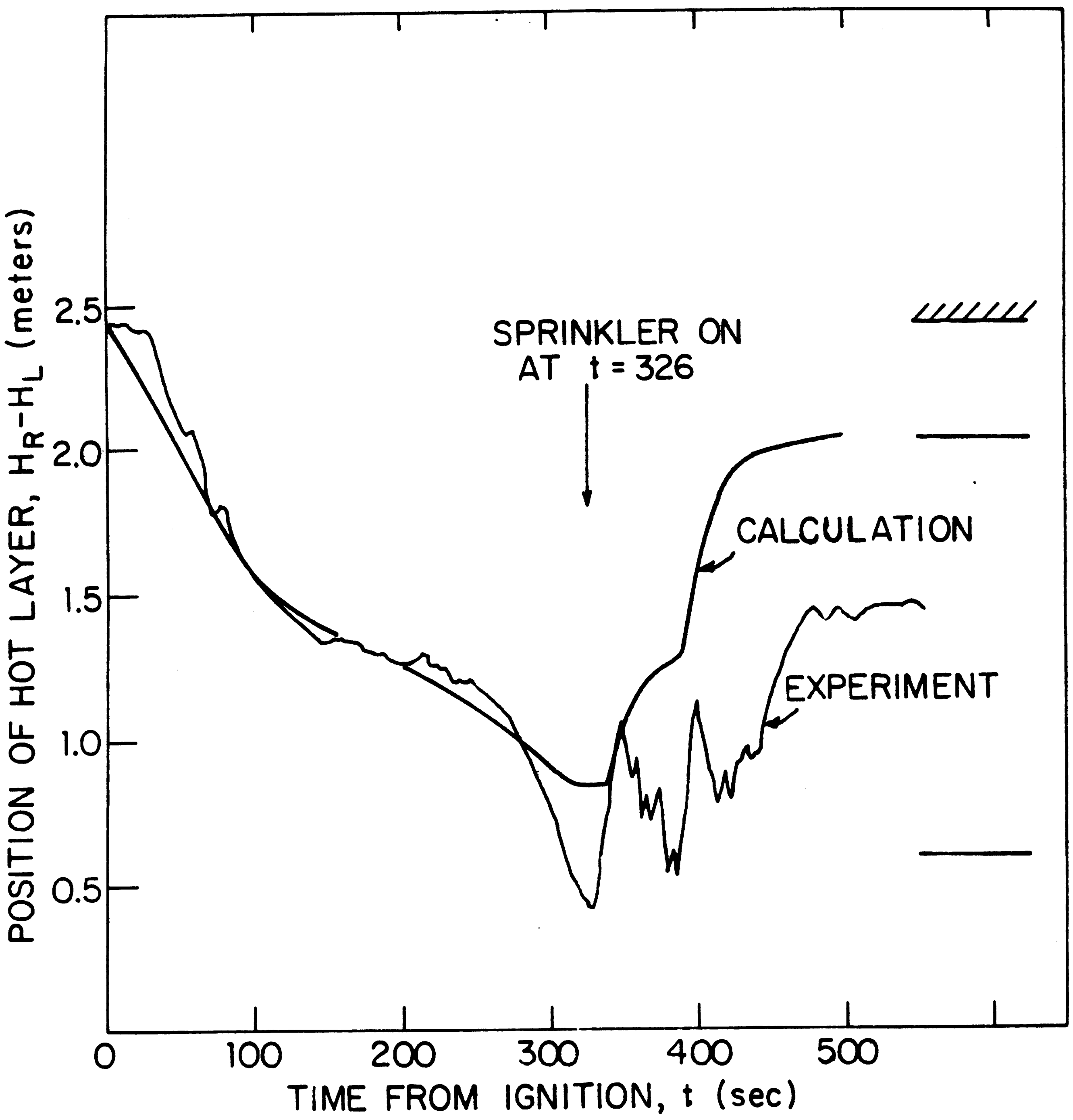


Fig. 39. "Experimental" vs predicted values of layer depth, h_L. (See discussion of Fig. 38 in text to understand why "experimental" is in quotes.)

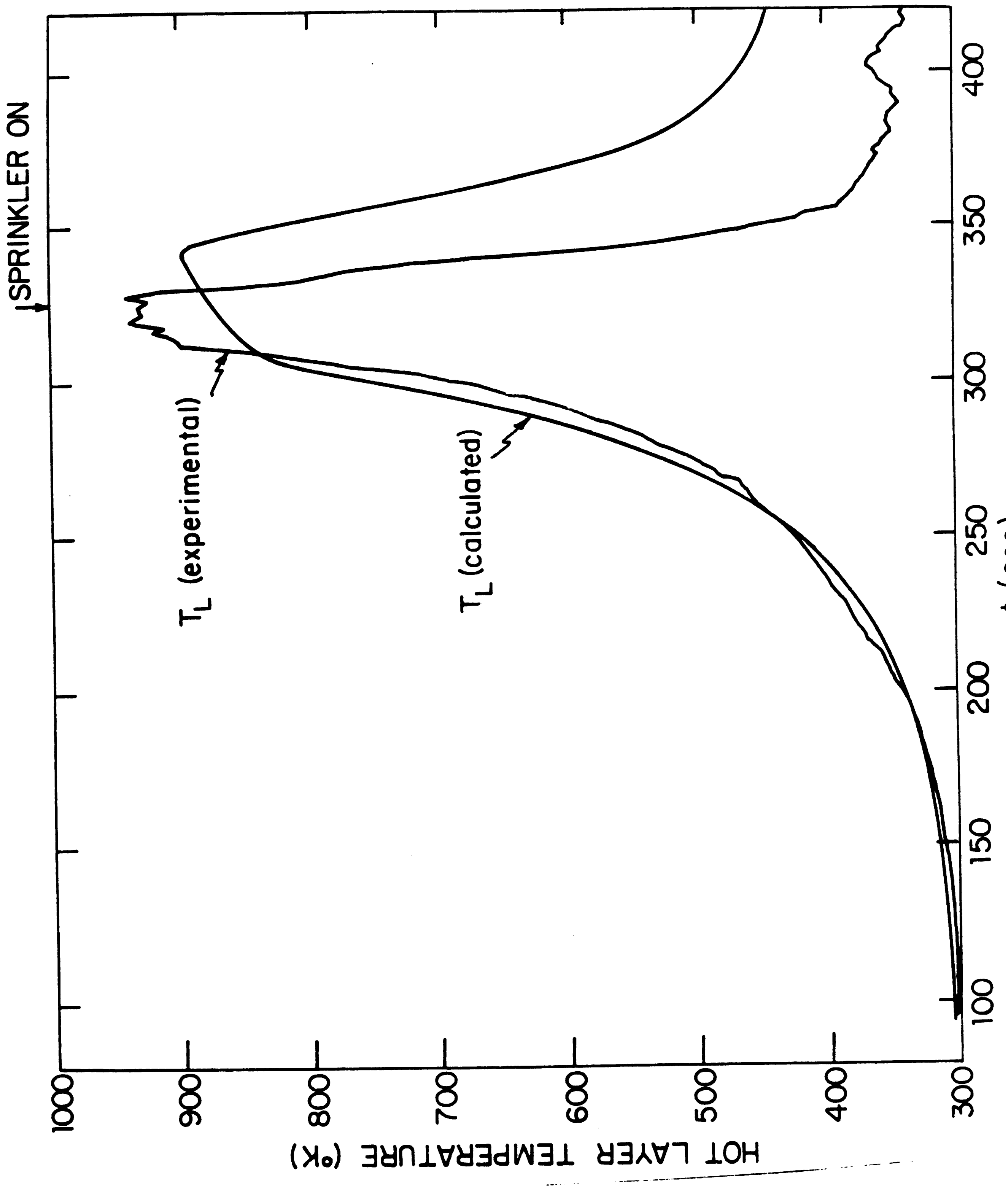


Fig. 40. "Experimental" <u>vs</u> calculated values of hot layer temperature, T_L.

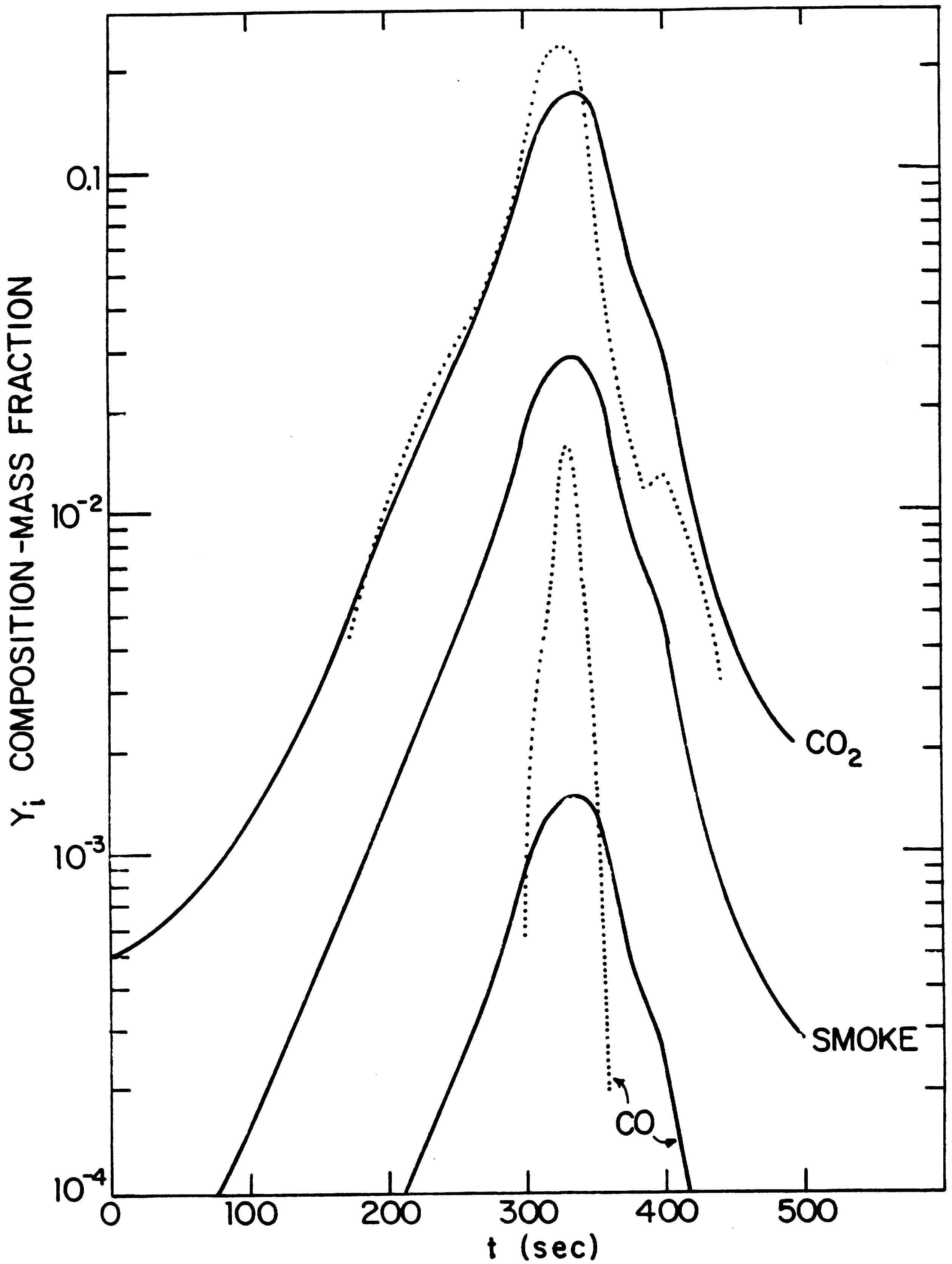


Fig. 41. Experimental vs predicted gas and particulate concentrations.

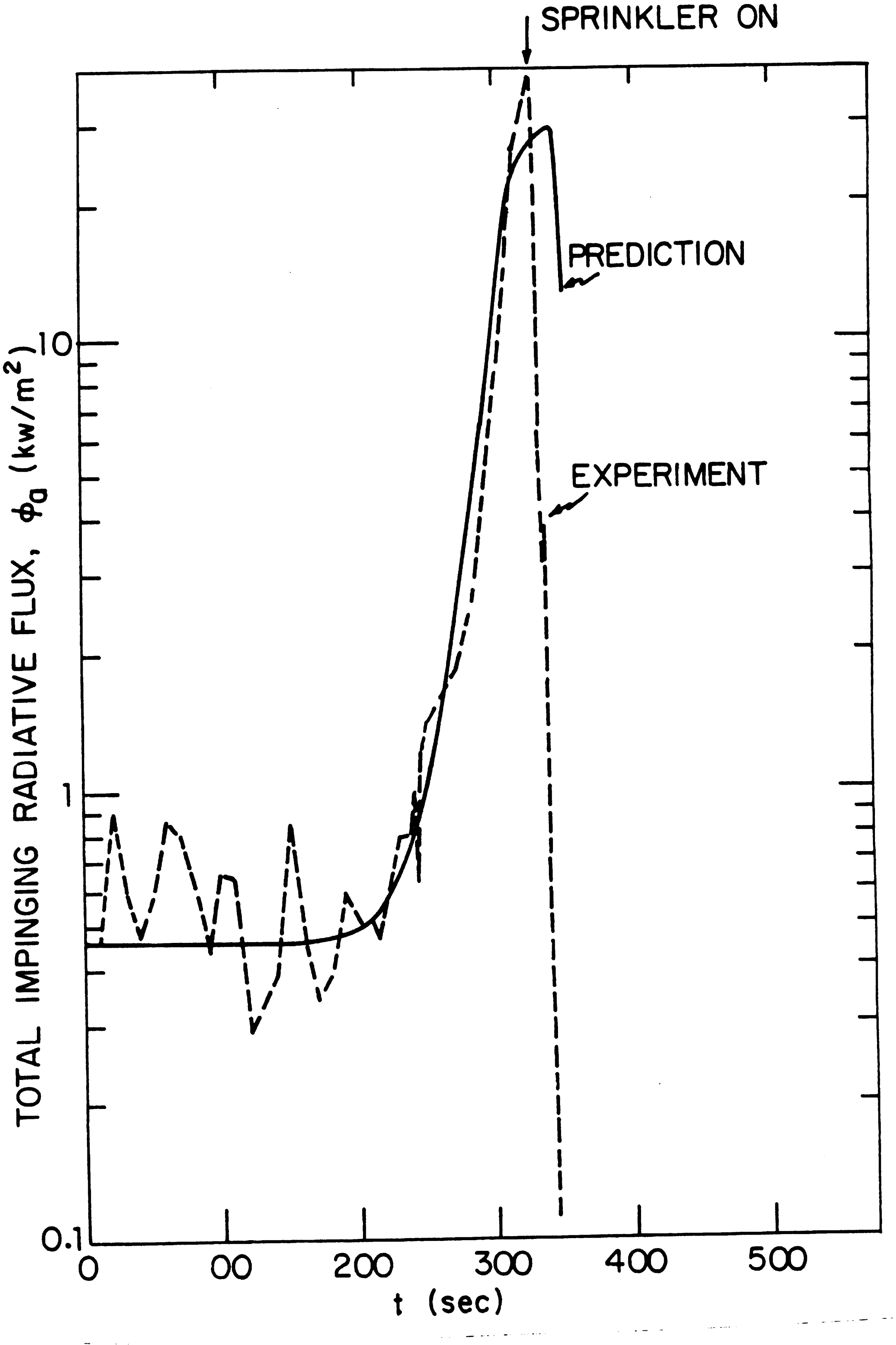


Fig. 42. Heat flux at center of floor (test #1, July 6).

VIII. USE OF THE PROGRAM.

The input and output.

As in the previous sections, I will first indicate very succinctly the input that is needed and the output available. Then I will go into details for those requiring a deeper understanding, starting with the explanatory notes.

Data which user must supply for a run(1).

Room(2): Length in the x direction(3) (in meters)

Length in the y direction

Height (in m)

Ambient temperature T_a (in degrees Kelvin)

Object(s): Number of objects (5 maximum -- i.e., $0 \leq N_o \leq 5$)

For each -- its mass (in kg)

Location of center of exposed surface (x, y, height)

Size (a x b and its thickness θ, or R and θ)(4)

and(5) R_{max}. For pools and burners, $R = R_{max}$.

Is it horizontal or vertical?

If vertical, what is the angle between the exposed surface and the x-z plane?

Thermophysical properties of materials:

Thermal conductivity	κ	(in j/sec m deg C)
Specific heat	c	(in j/kg ^oC)
Mass density	ρ	(in kg/m^3)
Heat of combustion	H_c	(in j/kg)

Is it flaming, or not? (6)

If yes, is it a growing fire, a pool fire, or a gas burner fire?

For pools and growing fires, need

Heat of vaporization H_v (in j/kg)

For growing fires, we also need the

Spread rate parameter A (in m/sec)

For burner fires, need

Gas flow rate $\dot{m}_{gas}$ (in kg/sec)

If no, we need

Ignition temperature T_{ig} (in ^oK)

Mass fractions f_i of CO, CO_2, smoke, and H_2O produced in combustion(7).

Combustion efficiency χ

Air-fuel mass ratio γ

Stoichiometric air/fuel mass ratio γ_s

Vent(s): Number of vents $(1 \leq N_v \leq 5)$

For each, its size (height, width) and its vertical location ("transom" height), h_t.

Wall: Thickness τ (in m)

Material parameters -- as under Objects.

Then the length of the run (in seconds) and the step size, Δt, in seconds.

Recall that the maximum number of variables permitted in this version is 100. Hence the user must make sure that $N \leq 100$, where N is given by eq. (1); the input for Mark 5 is not sophisticated enough to warn the user if he should inadvertently choose a mix involving too many variables -- e.g. five objects and three vents.

Notes:

(1) If the user does not specify the input values, then a default set will be chosen, corresponding to the standard run (see section VII). This is done in blocks of information -- that is, we might accept all of the default values except the set for the vent, which we change (to whatever we want).

(2) Even though the program was designed for a multiroom structure, it is still valid for one room only.

(3) If the room has only one door, then the lower left-hand corner of the room (as we face the wall with the door in it) is the origin of coordinates. The x-axis extends to the right, the y axis directly away, and the z-axis upward. This constitutes a right-handed Cartesian coordinate system. If the room has more than one door, the user must make some (consistent) choice.

(4) a and b are the dimensions of the (assumed rectangular) object in the x and y directions, respectively -- this assumes the object is lined up with the room, of course. R is its radius if the object is cylindrical. If it is rectangular, the program internally finds an equivalent radius -- i.e.,

$$\pi R^2 = ab.$$

(5) If the object can burn down its sides as well, then the total maximum exposed area includes the area of the sides, as well, and we

define--

for a rectangle, $\pi R^2_{max} = ab + 2(a+b)\theta$

for a cylinder, $\pi R^2_{max} = \pi R^2 + 2\pi R\theta$

where θ is the slab thickness.

(6) Note that we can start with several independent (simultaneous) fires.

(7) Because of the uptake of O_2, the "fractions" of CO_2, etc., might be greater than unity. Thus for the P.U. foam, the stoichiometric air/fuel mass ratio is 9.84; interpreting χ as completeness of combustion (for the sake of simplicity), then since $\chi \cong 0.65$, we have $m(O_2) \doteq 1.48m_f$, where $m(O_2)$ is the mass of oxygen which combines with the mass m_f of fuel vapors. Thus the sum of mass fractions must be 2.48.

Most of the input has a floating-point format -- that is, it must have a decimal point. This is important, as on some systems, if the user forgets to enter a decimal point, the machine will assume one exists (at a place determined by left-justification of the number), and accept the entry -- which will most likely be off by a few orders of magnitude! For a few items (H_c, H_v), it must be in exponential notation -- thus, for example, .00094 would have to be entered as 9.4E-4. As has been indicated above, the units are S.I.

Availability.

The user of the program will generally have it available on a tape, in a number of FORTRAN files. These must be compiled and linked at the user's facility. Once that's done, typing "RUN MARK5" will begin execution: requests for information will appear on the screen, which the user must supply, as indicated above. The program has been so written that striking the RETURN key will give the default option (this is not always true in BATCH mode). If this is not the case, or if an error is made, the question will be reasked.

When all the required information has been input, the calculation begins and proceeds to the end, or to where it runs into difficulties, at which point it will state "YOU ARE IN TROUBLE. DO YOU WANT TO CONTINUE?" The user has then to make a choice, and the correct choice will soon become evident. Once the run has been terminated (one way or the other), the computed information is in data file FIRE; at our facility, it is FIRE.DAT, and can be retrieved by typing "PRINT FIRE.DAT".

If another case is to be run, the user must again type "RUN MARK5", and begin all over again.

At our computing facility, we of course have a fully compiled and linked program on disk, and can run problems immediately. We also have Mark 4 on disk on a PDP 10 (at Aiken Computing Lab, nearby) and this is accessible to users directly *via* the ARPA net. However, it is not yet clear whether the latter facility will continue to be available, and hence Mark 5 will not be accessible *via* the ARPA net, at least for the present.

Running the program.

Execution begins with a request for "PROGRAMMER, RUN NUMBER" -- anything may be entered here; generally we write the programmer's initials followed by a number -- e.g. HM105. Then an opportunity to type in any comments at all about the run. Next is the apparently silly question: "IS THIS BATCH MODE?" (Unfortunately, there is an (elementary) bug in the program here, and if the response is "yes", then the user will very shortly run into a difficulty which will prevent execution.) Then, even for a standard run, there are eight more questions:

1. Do you want to change the physical subroutines?
2. Do you want a configuration other than the standard?
3. Output format?
4. Time increment between outputs to screen?
5. Ditto for output to disk data file?
6. Run length?
7. Basic time increment? (2 seconds is the default Δt)
8. Do you want to use the debugger?

Question 1 above is asked, because the user has a choice of subroutines for the calculation of the surface temperature of objects, and for the calculation of the absorptivity of the hot layer. Question 2 is self-explanatory, and the appropriate input has to be typed in, as listed at the beginning of this section. The output format referred to in question 3 is that to the disk (and hence to the printer), and the choices available are discussed in section IV.7.

For question 4, the default interval is 20 sec, for question 5 it is 10 sec. The default run length is 500 sec (Q. 6), and the basic time increment Δt is 2 sec (Q. 7). When the answer to the last question

(Q. 8) is "no", the program recapitulates the input for checking, then goes ahead and computes the fire.

The Debugger.

The debugger is available for programmers or skilled users for tracking down difficulties, should they appear. When the answer to question 8 above is "yes" (or <RET>), then eight further questions are asked:

When should the debugger cut in? How often should output go to the TTY thereafter (again, 20 sec is default)? How often to the disk? Should the message "leaving subroutine..." appear every time the calculation leaves a subroutine? (This option has very rarely been found to be useful, and will probably be dropped). Should output be displayed after each iteration? Do you want a list of the variables "in the system" at every rescaling? Or, do you want just those entering or leaving "the system" (i.e., ICOR) upon rescaling? Finally, do you want these questions reasked at a later time (and what time is that?).

Most of these questions, incidentally, are also in Mark 4, and are displayed in a partial protocol on page 32 of the Users' Guide..., ref. 28.

We have found that the most useful option is the one which gives the output after each iteration; this allows the calculations to be followed in detail, and errors or difficulties can often be tracked down this way.

An interesting use of the debugger is the following: if a run "hangs up" at some moment t_c, due to a numerical instability, sometimes it is possible to sidestep the trouble merely by invoking the debugger at a time earlier than t_c, which was skipped over. Thus, if $t_c = 308$ (say), and there were calculations made at (say) $t = 150$ and 152 (but not at $t = 151$), then invoking the debugger at 151 forces an extra calculation there; the resulting numerical differences thereafter might succeed in bypassing the trouble at 308. A similar trick is to change one of the parameters in the calculation slightly -- s physically insignificant change in room size, for example, might avoid the difficulty. Of course it is better not to have such difficulties arise in the first place, but some 5% of the time they do, nevertheless.

In order to examine various arrays or other calculations which are not displayed, such as the matrix elements of a Jacobian, it is necessary to have recourse to the more powerful debugging options generally available via the operating system.

Stoichiometric air/fuel mass ratio γ_s

Vent(s): Number of vents $(1 \leq N_v \leq 5)$

For each, its size (height, width) and its vertical location ("transom" height), h_t.

Wall: Thickness τ (in m)

Material parameters -- as under Objects.

Then the length of the run (in seconds) and the step size, Δt, in seconds.

Recall that the maximum number of variables permitted in this version is 100. Hence the user must make sure that $N \leq 100$, where N is given by eq. (1); the input for Mark 5 is not sophisticated enough to warn the user if he should inadvertently choose a mix involving too many variables -- e.g. five objects and three vents.

Notes:

(1) If the user does not specify the input values, then a default set will be chosen, corresponding to the standard run (see section VII). This is done in blocks of information -- that is, we might accept all of the default values except the set for the vent, which we change (to whatever we want).

(2) Even though the program was designed for a multiroom structure, it is still valid for one room only.

(3) If the room has only one door, then the lower left-hand corner of the room (as we face the wall with the door in it) is the origin of coordinates. The x-axis extends to the right, the y axis directly away, and the z-axis upward. This constitutes a right-handed Cartesian coordinate system. If the room has more than one door, the user must make some (consistent) choice.

(4) a and b are the dimensions of the (assumed rectangular) object in the x and y directions, respectively -- this assumes the object is lined up with the room, of course. R is its radius if the object is cylindrical. If it is rectangular, the program internally finds an equivalent radius -- i.e.,

$$\pi R^2 = ab.$$

(5) If the object can burn down its sides as well, then the total maximum exposed area includes the area of the sides, as well, and we

define--

for a rectangle, $\pi R_{max}^2 = ab + 2(a+b)\theta$

for a cylinder, $\pi R_{max}^2 = \pi R^2 + 2\pi R\theta$

where θ is the slab thickness.

(6) Note that we can start with several independent (simultaneous) fires.

(7) Because of the uptake of O_2, the "fractions" of CO_2, etc., might be greater than unity. Thus for the P.U. foam, the stoichiometric air/fuel mass ratio is 9.84; interpreting χ as completeness of combustion (for the sake of simplicity), then since $\chi \cong 0.65$, we have $m(O_2) \cong 1.48 m_f$, where $m(O_2)$ is the mass of oxygen which combines with the mass m_f of fuel vapors. Thus the sum of mass fractions must be 2.48.

Most of the input has a floating-point format -- that is, it must have a decimal point. This is important, as on some systems, if the user forgets to enter a decimal point, the machine will assume one exists (at a place determined by left-justification of the number), and accept the entry -- which will most likely be off by a few orders of magnitude! For a few items (H_c, H_v), it must be in exponential notation -- thus, for example, .00094 would have to be entered as 9.4E-4. As has been indicated above, the units are S.I.

Availability.

The user of the program will generally have it available on a tape, in a number of FORTRAN files. These must be compiled and linked at the user's facility. Once that's done, typing "RUN MARK5" will begin execution: requests for information will appear on the screen, which the user must supply, as indicated above. The program has been so written that striking the RETURN key will give the default option (this is not always true in BATCH mode). If this is not the case, or if an error is made, the question will be reasked.

When all the required information has been input, the calculation begins and proceeds to the end, or to where it runs into difficulties, at which point it will state "YOU ARE IN TROUBLE. DO YOU WANT TO CONTINUE?" The user has then to make a choice, and the correct choice will soon become evident. Once the run has been terminated (one way or the other), the computed information is in data file FIRE; at our facility, it is FIRE.DAT, and can be retrieved by typing "PRINT FIRE.DAT".

ACKNOWLEDGEMENTS

This work was supported in part by the National Bureau of Standards, under grant G7-9011, and the Division of Applied Sciences, Harvard University.

Many people have contributed to the success of this work. For Mark 5, however, two Harvard undergraduates have been especially valuable: Barry London and Joshua Gahm.

REFERENCES

(1) K. Kawagoe, "Fire Behavior in Rooms" (BRI Report No. 27), Building Research Institute, Tokyo, 1958.

(2) V. Babrauskas and R.B. Williamson, Fire and Materials, 2 No. 2(1978)39.

(3) H.E. Mitler, "The Physical Basis for the Harvard Computer Fire Code," Harvard University Home Fire Project T.R. #34 (1978).

(4) H.W. Emmons, "The Prediction of Fires in Buildings", in 17th International Symposium on Combustion, (1978) p. 1101.

(5) H.W. Emmons, C. MacArthur, and R. Pape, "The Status of Fire Modeling in the U.S. -- 1978", 4th U.S./Japan Cooperative Program on Natural Resources, Tokyo (1979).

(6) R. Pape, "Computer Simulation of Full-Scale Room Fire Experiments", IITRI (Illinois Institute of Technology Res. Inst.), Report J6414, (1978).

(7) J.B. Reeves, C.D. MacArthur, and J.F. Mayers, "Dayton Aircraft Cabin Fire Model", University of Dayton Res. Inst. Report FAA-RD-76-120, vols. I, II, and III (1978).

(8) T. Tanaka, "A Model on Fire Spread in Small Scale Buildings", Third U.S.-Japan Panel on Fire Research and Safety, Building Res. Inst., Tokyo, (1978).

(9) A.C. Ku, M.L. Doria, and J.R. Lloyd, "Numerical Modeling of Unsteady Buoyant Flows Generated by Fire in a Corridor", in 16th International Symposium on Combustion, (1976) p. 1373.

(10) J. Quintiere, "The Growth of Fire in Building Compartments", paper presented at the ASTM-NBS Symposium on Fire Standards and Safety (1976).

(11) T. Kubota and E.G. Zukoski, "A Computer Model for Fluid Dynamic Aspects of a Transient Fire in a Multi-Room Structure (3rd ed.)," Cal. Inst. of Tech., (1979).

(12) R.B. Williamson, "Coupling Deterministic and Stochastic Modelling to Unwanted Fire," presented at NBS Symposium for P.H. Thomas (1980).

(13) J.M. Chaix and J. Galant, "Modéle Numerique Stationnaire de la Propagation d'un feu en Compartement Ventilée"; Bertin et Cie (1979).

(14) U.S. Dept. of Commerce Report No. 003-000-00537-2: "Highlights of Fire in the U.S.", National Fire Prevention and Control Administration (NFPCA), National Fire Data Center.

(15) W.E. Becker et al, "Fire Research on Cellular Plastics : The Final Report of the PRC"(1980).

(16) A.C. Fernández-Pello & F.A. Williams, "A Theory of Laminar Flame Spread Over Flat Surfaces of Solid Combustibles", Combustion and Flame 28 (1977), p. 251.

(17) L. Orloff, private communications, 1977.

(18) H.W. Emmons, H.E. Mitler, L.N. Trefethen, "Computer Fire Code III," Home Fire Project Technical Report No. 25 (Harvard, 1978).

(19) J. de Ris, "Fire Radiation -- A Review", 17th Int. Combustion Symposium (1979), p. 1003.

(20) B.R. Morton, G.I. Taylor, and J.S. Turner, "Turbulent Gravitational Convection from Maintained and Instantaneous Sources," Proc. of Royal Soc. (London) A234 (1956), 1.

(21) A. Tewarsen, "Heat Release Rates from Burning Plastics," J. Fire and Flammability 8 (1977), 115; and in several technical reports for FMRC.

(22) A.T. Modak, "Radiation from Products of Combustion", Paper presented at 11th (fall) meeting of Eastern Section of Combustion Institute (1978).

(23) J.B. Scarborough, "Numerical Mathematical Analysis", 5th ed. (Johns Hopkins Press, 1962).

(24) G.C. Byrne & C.A. Hall, "Numerical Solution of Systems of Nonlinear Algebraic Equations", Academic Press (1973).

(25) G. Forsythe and C.B. Moler, "Computer Solutions of Linear Algebraic Systems", Prentice-Hall (1967).

(26) J.M. Bennett, "Triangular Factors of Modified Matrices", Numerische Mathematik 7, 217-221 (1965).

(27) J. Ramsdell, private communications.

(28) H.E. Mitler, "Users' Guide for the Harvard Computer Fire Code", Harvard University Home Fire Project Tech. Report #37 (1979).

(29) "Numerical Methods for Nonlinear Algebraic Equations", Gordon and Breach (1970), P. Rabinowitz, editor.

(30) B. Leckner, "Spectral and Total Emissivity of Water Vapor and Carbon Dioxide", Combustion and Flame 19 (1972) 33.

(31) J.D. Felske and C.L. Tien, "Calculation of the Emissivity of Luminous Flames", Combustion Science and Technology 7 (1973)25.

(32) R.L. Alpert et al, "Influence of Enclosures on Fire Growth, Vol. I: Test Data", FMRC Technical Report OAOR3.BU.0-7, (July-August 1977).

(33) H.E. Mitler, "Comparison Between Theory and Experiment for a Burning Room", Harvard University Home Fire Project Tech. Report #46 (1981).

(34) J.C. Nach, "Compact Numerical Methods for Computers", (J. Wiley, 1979).

(35) C.W. Gear, "Simultaneous Numerical Solutions of Differential-Algebraic Equations", IEEE Trans. Circuits Theory, Vol. CT-18 (Jan. 1971) 89.

(36) H.W. Emmons, "A Note on Minimizing the Unknowns for Computation of a Large System of Equations", Home Fire Project Tech. Report #26, Harvard University (1978).

(37) J. Ramsdell, "Finding Minimal Feedback Vertex Sets", Harvard Home Fire Project Tech. Report #47 (1981).

(38) H.E. Mitler and J. Rockett, "How Accurate is Mathematical Fire Modeling?" J. of the Georgian Academy of Sciences (to be published; in Russian). Also to be published in English as an NBS special publication.

APPENDICES

Appendix A. Notation.

It is usual practice to use Fortran symbols in ways suggesting common names and then to supply a dictionary of the symbols actually used. However, a fire in a large building has a large number of geometric, compositional, and physical properties. To avoid arbitrary selection of symbols for closely related things (eg. x coordinate of many objects) we have selected a naming system which uses each of the six Fortran places for preassigned purposes.

This results in labels which are not euphonious, nor universally satisfactory: not every variable we want to use can be described according to our scheme. This is principally due to the FORTRAN limitation to only six alphanumeric characters for a name; nevertheless, it has served us reasonably well. I can do no better than to reproduce (with some minor changes) the description given in CFC III:

The Fortran name of any quantity in COMMON will be selected as follows:

1 2 3 4 5 (6)

A. Position 1 is used for derivatives, with respect to:

Z	nothing	
T	time	$\frac{\partial}{\partial t}$, $(\dot{\ })$
X,Y,W	space	$\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$, $\frac{\partial}{\partial z}$, $(\ )'$
A	area	$\frac{\partial^2}{\partial x \partial y}$, $(\ )''$ per unit area
V	volume	$\frac{\partial^3}{\partial x \partial y \partial z}$, $(\ )''$ per unit volume
F	area and time	$\frac{\partial^3}{\partial t \partial x \partial y}$, $(\dot{\ })''$ -- i.e., a flux
S	space and time	$\frac{\partial^4}{\partial t \partial x \partial y \partial z}$, $(\dot{\ })''$ -- i.e., a source
D	difference Δ; exact definition to be specified by the user	

B. Position 2 is used for a physical quantity;

A area

B	width
C	specific heat
D	diffusion coefficient
E	energy
F	view factor
G	thermal diffusivity
H	height
I	radiant intensity
J	thermal conductivity
K	temperature
L	length
M	mass
N	thickness
O	heat transfer coefficient
P	pressure
Q	heat transferred
R	emissivity
S	coordinate
T	time
U	absorptivity
V	volume
W	z-coordinate
X	mole fraction (or x coordinate)
Y	mass fraction (or y coordinate)
Z	nothing (or z coordinate)

C. Position 3 specifies an object, place, or direction

A	ambient
D	down (lower layer, depth, mass flow, etc.)
F	fuel
L	hot layer
O	object (will include <u>all</u> objects, distinguished by subscripts)
P	plume
R	room
U	upper (layer, or depth, or mass flow, etc.)
V	vent
W	z-direction, or Wall

X	x-direction
Y	y-direction
Z	nothing

D. Position 4 specifies a composition variable.

B	charring plastic
C	cellulose
D	carbon dioxide
E	soot
G	combustible gas (usually reported as "hydrocarbon")
H	hydrogen
I	inert ingredients
J	char
M	carbon monoxide
N	nitrogen
O	oxygen
P	plastic
S	smoke
W	water (or H_2O vapor)
Z	none

E. Position 5 specifies special conditions.

A	ambient
B	boundary condition
C	critical condition (user defined)
D	convection
E	outside
F	final condition
I	initial
M	maximum value
N	minimum value
R	radiation
S	sum
Z	none

F. Position 6 is normally left blank. However, it is also used as a suffix for numerical handling:

Ø (zero) or P:	value at previous timestep
1	value on output from the current physical subroutine.

This notation has performed well except for one purpose, namely the transfer of heat or mass, in which it is desirable to indicate both the source and destination of the indicated quantity. Since most of these quantities are computed initially as fluxes we have made an exception to the above rules as follows:

If the first character is F (flux: F, in position 1) then the character in position 3 is the <u>source</u> of the indicated quantity while the character in position 4 is the <u>destination</u> of the indicated quantity.

For constants of nature (e.g. g), mathematical constants (e.g. π) and other universal quantities, the common symbol will be used (G,PI).

There are a number of input parameters (not variables) for which the naming convection has <u>not</u> been used; e.g. ZKOPY, the temperature at which pyrolysis begins, or ZKOIG, the temperature of ignition (for an object). These are included in Appendix B.

Since a large building will have many rooms and each room will contain many objects which heat, burn, produce thermal plumes, etc. most of the variables are subscripted, each over a range ($0-n_i$) selected by the user. In the numerical subprograms, in which all variables are scaled to order 1 and named routinely X(I), the index I is set by the computer in the range 0-N where N is the sum of all other ranges, i.e. $N = \sum_i n_i$.

This entire calculation in the physical subprograms is carried out in SI units. For those who wish to input or output data in other units, a special units conversion subprogram will eventually be developed. This has not yet been done.

The dictionary which follows will illustrate the above rules and will present all quantities so far used in CFC V. The dictionary also shows the subscript character (if any), the name of the subprogram which is responsible for generating values of that quantity, and the physical units in which it is given.

Dictionary B.4 also shows the subprogram names as currently used. They are generally 4-character names followed by two numbers which are assigned when the subprogram has been completely developed and put into the final listing. Thus there can be up to 99 versions of each subprogram, which seems like more than enough. (A subprogram developer should use the last two characters for his own notation to distinguish a program under development from those of proven value.)

The general Fortran notation is used for all input and output informatio
and all variables in common. However, <u>inside</u> of any subroutine any con-

venient symbol can be used without regard for <u>internal</u> symbols in any other subroutine, of course.

Appendix B. Dictionaries.

B.1: Variables. We list the variables and parameters, together with their symbol, the subroutine where each is calculated (or found), and a brief definition.

FORTRAN Symbol	Found in	Description
ALPHA = α	DATA	Plume entrainment coefficient (taken to be 0.1)
ANGH(KO) = θ_H	INPUT	Angle which (plane) surface of object KO makes with the horizontal (in degrees). At present, if this angle is input as $> 45^o$, it will be taken to be 90^o -- i.e., a vertical object; if $< 45^o$, it is taken to be 0^o -- i.e., a horizontal object.
ANGV(KO) = γ	INPUT	Angle which vertical surface of object KO makes with the plane xz, i.e., the wall running along the x-axis (in degrees). See fig. 27.
BETA = β	TMPO01	A convenient combination of parameters used in TMPO01: $\beta \equiv 1/\sqrt{\pi\rho c\kappa}$
CD = c_d	DATA	Vent flow coefficient (we use 0.68)
CHI(KO) = χ	INPUT	Fraction of combustion energy H_c actually released in open-air burning.
CP = c_p	DATA	Specific heat of air at STP (1004.0 joules/kg^{o}C).
DT = Δt	MAIN	Size of current timestep. This increment is in seconds.
DTINIT = Δt_o	INPUT	Initial step size in seconds. This will remain the maximum step size in the calculation.
EB(KO) = e_b	INPUT	Emissivity/absorptivity of surface of object KO (unless otherwise specified, we take e_b = 0.98).

FCO(KO) = f'_{CO}	INPUT	Mass fraction of (combustible part of) object KO evolved as CO, when it burns in the open.
FCO2(KO) = f'_{CO2}	INPUT	Mass fraction of object KO evolved as CO_2, when it burns.
FH2O(KO) = f'_{H2O}	INPUT	Mass fraction of object evolved as H_2O, when it burns.
FS(KO) = f'_s	INPUT	Mass fraction of object KO evolved as smoke (C and condensed hydrocarbons), when it burns.
FQLOR(KO) = $\dot{q}''_{LO}$ = ϕ_{LO}	RDNO, RNLO	Radiative flux to surface of object KO, from the hot layer in the enclosure (w/m^2).
FQLWD(KW,JSIDE) = $\dot{q}''_{LWD}, \dot{q}''_{AWD}$ or ϕ_{LWD}, ϕ_{AWD}	CNVW	Convective heat flux to side JSIDE of wall KW, from the hot layer (in w/m^2). Thus FQLWD(KW,1) is the flux from the inside of the room to the inner part of the wall/ceiling. FQLWD(KW,2) is the flux (from outside) to the other side of the extended ceiling.
FQLWR(KW,J) = ϕ_{LWR}, ϕ_{AWR}	RADW	Radiative flux to side J of wall KW, from the hot layer (w/m^2).
FQPOR(KO) = $\dot{q}''_{PO}$	RDNO, RNPO	Radiative flux to surface of object KO, from the flame(s) (w/m^2).
FQPP = ϕ_{ff}	RNFF	Flux from flame-flame interaction. Non-zero only if target is also burning.
FQPWR(KW, J) = $\sum_f \phi_{FW}$	RDNW	Radiative flux to side JSIDE of wall KW, from the flame(s) (w/m^2).
FQWOR(KO) = $\dot{q}''_{WO}$	RDNO, RNWO	Radiative flux to surface of object KO, from the hot walls and ceiling (watts/m^2).
FQPP(KO1,KO2) = ϕ_{12}	RNFF	When objects KO1 and KO2 are both flaming, this is the flux impinging on the base of KO2 due to absorption by its flame, of radiation from KO1's flame.
G	data	Acceleration due to gravity (9.8 m/sec^2).
PI = π	DATA BLOCK	π = 3.141592...

FORTRAN Symbol (cont'd)	Found in	Description
QF(KO) = H_c	INPUT	Heat of combustion of material making up object KO (in joules/kg).
QVAP(KO) = H_v	INPUT	Heat of vaporization (or pyrolysis) of KO_2 (in joules/kg).
R	DATA	Gas constant
SIGMA = σ	DATA BLOCK	Stefan-Boltzmann radiation constant (5.67×10^{-8} w/m^2 deg^4).
T = ZKFZZ	DATA	Preselected temperature of flaming gases (1260^oK).
TELZD(KR) = $\dot{E}_{LD}$	CNVL	Rate of increase of energy of the hot layer by convection in room KR (in watts) -- generally negative.
TELZR(KR) = $\dot{Q}$	RDNL	Rate of change of energy of the hot layer in room KR, by radiation (watts).
TELZZ(KR) = $\dot{E}_L$	LAYR	Net rate of change of the energy of the hot layer in room KR (in watts).
TEOZZ(KO) = $\dot{E}_f$	FIRE $\left\{\begin{matrix}\text{GFIR}\\\text{PFIR}\\\text{BFIR}\end{matrix}\right.$	(Negative of) chemical energy output rate (in watts) of object KO, when it is burning.
TEPZR(KO) = $\dot{E}_{PR}$	RDNP	Radiative power loss from flame above (burning) object KO (in watts).
TEPZZ(KO) = $\dot{E}_P$	PLUM	Energy flow out the top of plume KO (in watts).
TEUZZ(KV) = $\dot{E}_u$	VENT	Energy convected out the upper part of vent KV (in watts).
TMDZZ(KV) = $\dot{m}_u$	VENT	Mass outflow in the lower part of vent KV. (A negative value means outflow.) In kg/sec.
TMFGZ(KO) = $\dot{m}_g$	INPUT	Gas flow rate, when object KO is a gas burner (in kg/sec).
TMLZZ(KR) = $\dot{m}_L$	LAYR	Rate of increase of mass of the hot layer in room KR. In kg/sec.

FORTRAN Symbol (cont'd)	Found in	Description
TMOZZ(KO) = $\dot{m}_f$	FIRE {GFIR, PFIR, BFIR}	Rate of change of mass of object KO (kg/sec) -- generally negative, of course.
TMPLU(KO) = $\dot{m}'_e$	PLUM	Rate of entrainment of layer gases by the upper part of plume KO (kg/sec) -- i.e., the part in the hot layer. In CFC V, $\dot{m}'_e$ is taken to be zero.
TMPZZ(KO) = $\dot{m}_p$	PLUM	Mass flow out the top of the plume over object KO (into layer) -- in kg/sec.
TMRZZ(KR) = $\dot{m}$	VENT	Net mass (out-)flow rate from room KR.
TMUZZ(KV) = $\dot{m}_u$	VENT	Mass flow out the upper part of vent KV (kg/sec).
PSI(KO) = ψ	FIRE	Semiapex angle of cone modeling the flame over burning object KO (in degrees).
TPSI(KO) = tan ψ	FIRE {GFIR, PFIR, BFIR}	Tangent of ψ; actually, ψ is not carried as a variable -- only TPSI.
VMAZZ = ρ_a	DATA BLOCK	Density of ambient air (1.177 kg/m^3 at 300^{o}K).
VMLZZ(KR) = ρ_L	LAYR	Density of hot layer in room KR (in kg/m^3).
VMOZZ(KO) = ρ_i	INPUT	Density of object KO (in kg/m^3).
VMWZZ(KW) = ρ_W	INPUT	Density of wall KW (in kg/m^3).
XLAMDA = λ	DATA	Radiation mean free path (in m).
XGAMMA(KO) = γ	INPUT	Air/fuel mass ratio during open-air burning; calculated to be 14.45 for P.U. foam.
XGAMAS(KO) = γ_s	INPUT	Stiochiometric air/fuel mass ratio. Calculated to be 9.85 for P.U. foam.
ZBVZZ(KV) = B	INPUT	Width of vent KV (in meters).
ZCAZZ = c_p	DATA	Specific heat of ambient air (1004 j/kg deg).

FORTRAN Symbol (cont'd)	Found in	Description
ZCFZZ(KO) = c_g	INPUT	Specific heat of pyrolysis gases from object KO, when it's burning (in joules/kg deg C).
ZCOZZ(KO) = c_i	INPUT	Specific heat of object KO (j/kg deg).
ZCWZZ(KW) = c_W	INPUT	Specific heat of wall KW (in j/kg deg).
ZELZZ(KR) = E_L	LAYR	Energy of the hot layer in room KR (in joules).
ZGOZZ(KO) = α	DATA	Thermal diffusivity of object KO (in m^2/sec).
ZGWZZ(KW) = α_W	DATA	Thermal diffusivity of wall KW (m^2/sec).
ZHBZZ(KV) = h_b	INPUT	Height of vent sill above floor (in meters).
ZHLZZ(KR) = h_L	LAYR	Depth of the hot layer in room KR (in m).
ZHOZZ(KO) = h_o,h_f	INPUT	Height of surface of object KO above floor (in meters).
ZHPZZ(KO) = h_p	PLUM	Height of plume KO (between hot surface and layer interface; in meters).
ZHRZZ(KR) = h_R	INPUT	Height of room KR (in m).
ZHTZZ(KV) = h_t	INPUT	Distance of top of vent KV below ceiling -- i.e. transom height (in m).
ZHVZZ(KV) = h_v	INPUT	Height of vent KV (m).
ZJOZZ(KO) = k	INPUT	Thermal conductivity of object KO (w/m deg K).
ZJWZZ(KW) = k_W	INPUT	Thermal conductivity of wall KW (w/m deg K).
ZKAZZ = T_a	DATA	Temperature of ambient air (in degrees Kelvin; default value is 300 ^oK).
ZKFZZ = T	see T	
ZKLZZ(KR) = T_L	LAYR	Temperature of the hot layer in room KR (^oK)

FORTRAN Symbol (cont'd)	Found in	Description
ZKOZZ(KO) = T_s	TMPO	Temperature of the surface of object KO (^oK).
ZKOIG(KO) = T_{ig}	INPUT	Temperature of ignition of object KO (^oK).
ZKOPY(KO) = T_p	INPUT	Temperature at which object KO begins to pyrolyze (^oK).
ZKWZZ(KW,JSIDE)	TMPW	Temperature of side JSIDE of wall KW (in ^oK).
ZLAMDA(KO) = λ	DATA BLOCK	Radiation mean free path of photons in flame gases (i.e., absorption length -- in m). This is just the reciprocal of κ = ZUFZZ.
ZLRZX(KR) = L_x	INPUT	Dimension of room KR in x-direction (in m).
ZLRZY(KR) = L_y	INPUT	Dimension of room KR in y-direction (in m).
ZMLZZ(KR) = m_L	LAYR	Mass of hot layer in room KR (in kg).
ZMOZO(KO) = m_o	INPUT	Original mass of pyrolyazble part of material in object KO (kg).
ZMOZZ(KO) = m_f	FIRE	Mass of object KO as a function of time (kg) -- i.e., the remaining mass.
ZNOZZ(KO) = θ	INPUT	Thickness of object KO (meters)
ZNWZZ(KW) = τ	INPUT	Thickness of wall KW (m)
ZOAZN = h_e	DATA	Minimum heat transfer coefficient of (quiescent, ambient) air (taken to be 5 w/m^2 deg K).
ZOAZZ = h	DATA	Heat transfer coefficient of non-quiescent air. We take it to be 10 w/m^2 deg C.
ZOLZZ(KR) = h_i	CNVW	Heat transfer coefficient of hot layer gases (to ceiling/walls) in room KR (in w/m^2 K).
ZOLZM = a	DATA	Maximum heat transfer coefficient of hot layer gases (in w/m^2 ^{o} C). We take it to be 50 w/m^2 ^{o} K).

FORTRAN Symbol (cont'd)	Found in	Description
ZOLZN	DATA	Minimum heat transfer coefficient of hot layer gases -- we take it to be the same as ZOAZN -- i.e., 5 $w/m^{2\circ}K$.
ZPAZZ = $p_a \equiv 0$		Ambient air pressure, taken as reference pressure (in meters of air; 1 atm. = 8780m).
ZPRZZ(KR) = p_f	VENT	Pressure at center of floor of room KR, referred to p_a (in meters of air equivalent).
ZRFZI(KO) = R_i	INPUT	Radius of object KO, in meters (see description of R_o in subroutine RNPO).
ZRFZM(KO) = R_m	INPUT	Augmented radius of object KO -- i.e., maximum equivalent possible radius of fire, in m (see discussion under GFIR).
ZRFZO(KO) = R_o	INPUT	Initial radius of fire on object KO, in m (default value is .037 m).
ZRFZZ(KO) = R	FIRE	Radius of fire on object KO (in m).
ZTIG(KO) = t_{ig}	FIRE	Time of ignition of object KO (in sec).
ZTPYR(KO) = t_{py}	FIRE	Time of start of pyrolysis of object KO (in sec).
ZTZZZ = t	MAIN	Time from ignition (in sec).
ZUFZZ = κ_f	DATA	Absorption coefficient of flame gases (default value is 1.55 m^{-1}; L. Orloff).
ZULZZ(KR) = κ	ABSRB	Absorption coefficient of hot layer (in m^{-1}).
ZXOZZ(KO) = x_o	DATA	X coordinate of object KO, with respect to the local coordinate system of the room it is in (in meters).
ZYLDZ(KR) = Y_{CO2}	LAYR	Mass concentration of CO_2 in hot layer in room KR (in gm/gm).
ZYLMZ(KR) = Y_{CO}	LAYR	Concentration of CO in hot layer in room KR (in kg/kg).

FORTRAN Symbol (cont'd)	Found in	Description
ZYLOZ(KR) = Y_O	LAYR	Oxygen mass concentration in layer in room KR.
ZYLSZ(KR) = Y_S	LAYR	Concentration (by mass) of smoke (particulates, mainly C, and condensed hydrocarbons) in the hot layer in room KR.
ZYLWZ(KR) = Y_{H_2O}	LAYR	Mass fraction of H_2O in the hot layer in room KR.
ZYOZZ(KO) = y_o	DATA	Y coordinate of object KO (in room KR), in meters.

Appendix B.2 -- Subscripts.

Subscript	Purpose	Present Range
J, or JSIDE	Identifies the side of the wall under consideration: 1. inside room 2. outside of room (open air).	
KO	Identifies the object	5
KR	Identifies the room (KR = 1 only)	
KV	Identifies the vent	5
KW	Identifies the wall/ceiling (as with the room, there is only one in Mark5, though it has been DIMENSIONED to 5)	

Appendix B.3 -- Some flags and other variables.

Most of the indices appearing in the program are carried in one or another of the commons, and hence can be found on pp. 71-79. Those which are not in common appear here:

Symbol	Remarks
HALVE	Logical variable in subroutine NUMER. When .TRUE., time interval Δt must be cut in two.
HH	Fractional increment of variable whose influence coefficient is being calculated. At present HH = 0.001 of scaled variables.
ICOUNT	Two meanings: first, it gives the number of state changes which occurred in a given timestep. In subroutine SETJ, on the other hand, it is the number of the variable in JBLK (hence, varies from 1 to NVAR See pp. 69-70.)
ICONV	Flag to show convergence
	0 ==> not converged.
	1 ==> converged (to accuracy TOLER).
ICTRL	Index used in MAPS, to indicate whether a mapping packs a large, sparse array into a smaller, dense one, or vice-versa. (See fig. 19, p.6).
	0 ==> pack
	1 ==> unpack
INUM	Dummy index used only in SETI. It is the position of a variable in ICOR. (See fig. 23, p. 70.)
MET1	Index which indicates that a switch in numerical method is called for
	0 ==> Switch to JACB
	1 ==> Switch to NWSF
	2 ==> Switch to NWTN
NCONV	Index in subroutine CONV, indicating status of calculation (see p. 68):
	1 ==> converged
	0 ==> not yet converged; insufficient information available from fnorms to decide whether or not it <u>will</u> converge.
	-1 ==> diverging; halve the time increment (or change numerical method).
	-2 ==> diverging, and it is probably useless to halve Δt.
	m(m>1) ==> converging, and m is the expected number of iterations still needed.

Symbol (cont'd)	Remarks
REMAP	Logical variable in NUMER. When .TRUE., ICOR must be reset.
TOLER	Convergence criterion for scaled variables. At present ε_c=TOLER=3 x 10^{-4}.

Appendix B.4: Subroutines other than for physics.

Subroutine	See Page	Description
ALTINP		Called by INIT. In file I. Reassigns user I/O logical units to something other than the terminal -- used for batch mode. Questions which normally go to the TTY are now dumped into a default "garbage" file FOR000.DAT. However, not everything which goes to the TTY in interactive mode is written into this garbage file; to save I/O overhead, some WRITEs are ignored if the IBATCH flag is set (see INPUT3, DISP, VERIFY, and WRIT). In addition, to reduce the size of a batch deck of cards, the "verify" question from INPUT3 and VERIFY is suppressed. Input to ALTINP should be the logical unit number for the new input device, or a number N for a default FORTRAN file for input, known as FOR00N.DAT, with N $\neq$ 0.
CALS	63, 80	("Call Subs"). Called by MAIN, EXTRAP, JACB. In file S. Calls the physical subroutines in succession to compute physical variables. XK is the input vector and XK1 is the output vector. (Dummy subroutines TMPF, FHET, and PLHT have been inserted to assure that certain variables are not set equal to zero.) CALS (and before that, CLSB) was changed so as to implement the Gauss-Seidel method, in place of the Jacobi form of the successive substitution method. It now also allows for a variable subroutine-calling order via the array IORDER.
CALS1	80	Called by NWTN. In file S. This subroutine differs from CALS in that it leaves /VAR/ undisturbed. The new values are placed into the common block /NEWVAR/ as they are returned from the physical subroutines. This subroutine is used by NWTN to compute the Jacobian matrix.
CONV		Called by JACB, NWTN. In file N. This subroutine examines XK and XK1, and decides whether the array XK has converged. (Only considers variables in

Subroutine	See Page	Description
(CONV, cont'd)		the system -- i.e., in ICOR.) The result is given via index NCONV (which see).
COPINP		Called by INPUT3. In file I. Copies the input data from another section, when desired (e.g. the physical properties of object #3 might be identical to those of #2, but it is placed in another spot in the room). This simplifies data input.
DEBUG	81	Called by MAIN, TIGC, CALS. In file W. This routine contains an initialization section (entered when IFLAG=1) and a section that gives help to a user in trouble. To invoke the help section simply set IHELP, a variable in COMMON /DBUG/, to 1. This will force a call to DEBUG at the first possible moment, usually at the end of the current iteration.
DECOMP		Called by MSLV. In file N. Decomposes any matrix -- in particular, the Jacobian -- into a product of a lower-diagonal matrix L with unit diagonal entries and an upper-diagonal matrix U. The output matrices L and U are both **stored** in UL.
DELTAT	52, 60	Called by MAIN. In file M. Calculates Δt for next step. It does this by referring to the flags IIGNT, IOUTP, TCONV, DTOLD, and DTINIT (see common /CONTRL/).
DISP		Called by INPUT3. In file I. Displays input values as they are fed in, in blocks. Changes or corrections are easy to implement here.
EXTRAP	52, 56	Called by MAIN. In file M. Extrapolates XK, so as to come up with an initial guess at the converged values of the physical variables at a new timestep. Quadratic extrapolation is used whenever possible (when in Gauss-Seidel), linear extrapolation otherwise. Linear extrapolation is also used when in NWTN.
FIRE		(see MAIN)
INIT		Called by MAIN. In file I. Initializes variables for use in CFC (except for system variables initialized locally in subroutines). Part of input package. Calls INPUT3 to get data from the terminal.

Subroutine (cont'd)	See Page	Description
INPUT3		Called by INIT. In file I. Reads input data from the terminal.
JACB	18, 67	Called by NUMER. In file N. Implements the successive substitution method to solve the system of equations at a fixed timestep. Flag ICONV is set to 1 on exit if a converged solution has been obtained, to 0 if no solution has been obtained. If no solution has been attained, the time increment is halved. When there is no convergence at DT=.125 seconds, it switches over to the Newton method.
LIST		Called by SETI. In file W. This subroutine prints an alphanumeric list of variables in the system if the user has chosen that debugging option. If ICTR = 1, print a list of all variables in the system; If ICTR = 2, print a list of all variables entering or leaving the system.
LOOKUP		Called by LIST, NWSF, NWTN, MSLV, WRITØ3. In file W. This subroutine finds the name of the variable which is at (any) prescribed position of a variable array. It returns the stem of a variable name and its suffixes. Index is a position in the common block /VAR/. A sample calling sequence is `Call LOOKUP(JBLK(IMAX),ISTEM,ISUFF1,ISUFF2)` ISTEM should be dimensioned (5). To write the variable name out: `WRITE (IWTTY,200)(ISTEM(I), I=1,5),ISUFF1,ISUFF2` `200 FORMAT (1X,5A1,'(',I1,',',I1,')')` Currently, this subroutine is dimensioned so as to handle up to 20 room, object, or vent variables, and up to 10 wall variables. OFFSET contains the locations in /VAR/ preceding the first room, object, vent, and wall variable.
MAIN (or FIRE)	55	In file M. The control program of the Computer Fire Code.
MAPS	63,66, 73	Called by MAIN, DEBUG, WRIT, CALS, NWSTAT. In file S. This subroutine maps between the set of variables which actually exist in this run of the program and between the largest possible set of variables, via JCOR. ICTRL = 0 maps from XBIG to X. ICTRL = 1 maps from X to XBIG.

Subroutine (cont'd)	See Page	Description
MSLV		Called by NWTN. In file N. Directs solution of the linearized system of equations (23) by calling DECOMP and SOLVE. That is, it calls the Forsythe-Moler package to solve the rank-N linear system XJCB*X=B. Matrix XJCB is not in the calling sequence but in common block /NUMERC/. Parameter IDCMP controls the L-U decomposition of matrix XJCB into UL as follows: IDCMP=0: no new decomposition. IDCMP=1: call to "MODJ" to update decomposition for one new column(was removed on 6/12/80.) IDCMP=2: call "DECOMP" for complete new decomposition.
NUMER	52, 57	Called by MAIN. In file M. Controls calling of numerical methods; decides whether or not to halve Δt, and whether or not to reset ICOR.
NWSTAT	59, 62	Called by MAIN. In file D. Notes whether any objects have changed state, and calculates when this happened.

NWTN, NWSF	25, 65, 68	Called by NUMER. In file N. These implement a multivariate form of Newton's method to solve the system of equations at a fixed timestep.
		ICTRL = 1: Super Fast Newton Method (NWSF)
		ICTRL = 2: Slow Newton Method (NWTN)
		
		In NWTN, the entire Jacobian is updated at every iteration. This subroutine attempts to ensure convergence at the expense of efficiency. The increment HH formerly used in calculating the Jacobian has been replaced by 0.001 x the value of the variable itself. In NWSF, the Jacobian is kept constant.

RECAP		Called by INIT. In file I. Outputs a recapitulation of the input data, when that is all in and corrected; it is printed and displayed on the terminal as well.
RESETI	58, 61	Called by MAIN. In file M. Determines whether the program should recompute the set of variables in the system at this timestep. ICOR is recomputed if a variable becomes bigger than VMIN, if NSCAL seconds have passed since the last recomputation, and on the first 10 timesteps of the program.
SELSUB	58	Called by INIT. In file I. Allows user to select which version of some physics subroutines he wishes to use. Limited to TMPO and ABSRB, in Mark5.

Subroutine (cont'd)	See Page	Description
SETI	63, 70	Called by MAIN. In file S. This subroutine sets up ICOR, which maps from the set of variables in the system (i.e. to be used in Jacobian matrix and to be checked for convergence) to the set of all defined variables. It determines which variables are to be in the system. ICOR is now part of JBLK, lying in it above JCOR. IPTR(2,1) points to the bottom of ICOR. IPTR(2,2) points to the top if ICOR. INSYS and INSYSP is an alternate method of keeping track of which variables are in the system, now and at the previous call of SETI. INSYS and INSYSP correspond in size to JCOR, and equal 1 for variables in the system, 0 for variables not in the system .

SETJ	63, 69	Called by INIT. In file S. Generalized JCOR-initializing routine. Common /CCOR/, where JCOR and ICOR were located, has been expanded to one monolithic block of a DIM(600) array. JCOR and ICOR would be allocated within the block; JCOR at initialization time, ICOR continuously (at every rescaling its size may change).
		
		The array IPTR contains all the information about the boundaries of JCOR and ICOR:
		IPTR(1,1) = lower bound of JCOR;
		IPTR(1,2) = upper bound of JCOR;
		IPTR(2,1) = lower bound of ICOR;
		IPTR(2,2) = upper bound of ICOR;
		
		The array INVAR gives the number of variables associated with each component of the building: INVAR(1) - with a room, INVAR(2) - with an object, INVAR(3) - with a vent, INVAR(4) - with a wall.

SING		Called by DECOMP. In file N. Prints diagnostic messages relating to the solution of the system of linear equations. In particular, when the Jacobian is singular.
SOLVE		Called by MSLV. In file N. Solves eq. (23) by Gaussian elimination. That is, it solves the matrix equation A*X=B for vector X, where A is the product of the lower- and upper-diagonal factors stored in matrix UL.
VERIFY		Called by INPUT3, INIT. In file I. Asks the user whether he wishes to change (correct) the last entry he's made. If so, the last question is reasked, and a changed input can be made.

Subroutine (cont'd)	See Page	Description
WRIT	81	Called by MAIN, DEBUG, NWTN, JACB. In file W. Writes the input vector X onto device IDEVIC (teletype or disk). The pointer information in common block /POINTR/ determines which elements of the arrays in X are to be output.
WRITØ3	81	Called by WRIT, INIT. In file W. Provides the short-form output: Outputs 9 columns of information at user-chosen time intervals: time and 8 variables which the user chooses at the time of initialization (when called by INIT), or the default set. Note: intended for output with 120 or more columns only.

Appendix C. Sample Output.

On this page we display the standard ("long form") output of the program, for one time step. On the following three pages, we display the entire output for a standard run (S.R.), using the "short form" output, instead. In both cases, we first get a recapitulation of the input (not shown on this page, of course). The files used for run M240 (p. 178ff) antedate MARK 5 a little bit, and also include some utility programs (RTIME, TIMER) which are unique to our facility, and which serve to give us the CPU time, the date, etc. For the output, we chose to display ZULZZ (κ) rather than the "default" variable, ZKWZZ (T_W) to demonstrate that that can be done; the rest of the variables are the "default" set -- i.e., the set obtained when the variables are not explicitly specified.

```
VENT= 1:      TEUZZ= 2.2958E+05   TMJZZ= 4.3409E-01   TMOZZ=-4.2269E-
WALL= 1,1:    FOLWR= 1.0409E+03   FOPWR= 9.1181E+02   FOLWD= 2.5463E+
              ZKWZZ= 4.7584E+02
WALL= 1,2:    FOLWR= 0.0000E+00   FOPWR= 0.0000E+00   FOLWD=-3.2231E+
              ZKWZZ= 3.0645E+02
---------------------------------------------------------------------
---------------------------------------------------------------------
T= 460.000       DT= 2.000      NT= 233      NIT= 1504      TI= 5     3
ROOM= 1:      TELZR=-8.8694E+03   TELZD=-5.4011E+04   ZMLZZ= 6.2982E+
              TMLZZ=-4.9935E-04   ZELZZ= 3.3335E+06   TELZZ=-1.0823E+
              ZHLZZ= 1.0543E+00   ZKLZZ= 5.2717E+02   ZYLDZ= 1.6843E-
              ZYLOZ= 4.1913E-02   ZYLMZ= 3.9057E-04   ZYLSZ= 8.8193E-
              ZYLWZ= 1.4804E-02   ZPRZZ=-1.2448E-02
  OBJ= 1:     FOLOR= 6.1813E+02   FONOR= 1.1924E+03   FOPOR= 4.9002E+
  (ID= 1)     ZKOZZ= 7.2700E+02   ZMOZZ= 3.5056E+01   TMOZZ=-1.0927E-
              TEOZZ=-2.1681E+05
              ZHPZZ= 4.6909E-01   TMPZZ= 4.3363E-01   TEPZZ= 2.9265E+
              TEPZR= 5.8060E+04
              ZRFZZ= 3.6150E-01
  OBJ= 2:     FOLOR= 7.5165E+02   FONOR= 1.5547E+03   FOPOR= 1.6368E+
  (ID= 2)     ZKOZZ= 3.9044E+02   ZMOZZ= 1.0963E+00   TMOZZ= 0.0000E+
              TEOZZ= 0.0000E+00
VENT= 1:      TEUZZ= 2.2978E+05   TMJZZ= 4.3413E-01   TMOZZ=-4.2274E-
WALL= 1,1:    FOLWR= 1.0441E+03   FOPWR= 9.1185E+02   FOLWD= 2.5382E+
              ZKWZZ= 4.7641E+02
WALL= 1,2:    FOLWR= 0.0000E+00   FOPWR= 0.0000E+00   FOLWD=-3.4488E+
              ZKWZZ= 3.0690E+02
---------------------------------------------------------------------
---------------------------------------------------------------------
```

```
RUN M 240                              4-MAR-81          17:56:50
---------

S.R.  [illegible] → ( k(t), [illegible] for wall temp )


FILES IN USE:
  M10100,I10100,N10100,W10100,S10100,R10100,C10100,F10100,D10100,L10100,-
  P10100,T10100,V02021,RTIME,TIMER
PHYSICAL SUBROUTINES IN USE (ONLY FOR THOSE SUBROUTINES WITH MORE
THAN ONE VERSION AVAILABLE):
TMPD, VERSION   2
ABSRB, VERSION   2

         GEOMETRIC AND PHYSICAL PARAMETERS:

 ROOM NUMBER 1:
      LENGTH ALONG X=   2.4384
      LENGTH ALONG Y=   3.6576
      HEIGHT=   2.4384
      AMBIENT TEMPERATURE= 300.0
    OBJECT NUMBER 1 (ID= 1) :
      X-COORD= 0.8400    Y-COORD= 2.8180       HEIGHT= 0.6100
      ANGLE WITH HORIZONTAL=  0.00        ANGLE WITH VERTICAL X-PLANE=  0.00
      THICKNESS= 0.1000                   THERMAL CONDUCTIVITY= 0.0540
      SPECIFIC HEAT= 1900.                DENSITY= 48.00
      EMISSIVITY= 0.98                    CHI(FRACTION OF HEAT RELEASED)= 0.65
      HEAT OF COMBUSTION= 2.870E+07       HEAT OF VAPORIZATION= 2.054E+06
      PYROLIZATION TEMP= 600.0            IGNITION TEMP= 727.0
      AIR/FUEL MASS RATIO= 14.45          STOCHIOMETRIC MASS RATIO=  9.85
      INITIAL MASS= 6.8520                INITIAL RADIUS= 0.0370
      MAXIMUM RADIUS= 0.9677              OBJECT RADIUS= 0.8598
      FCO2(CO2 MASS/FUEL MASS)=1.504      FCO(CO MASS/FUEL MASS)=0.013
      FS(SMOKE MASS/FUEL MASS)=0.241      FH2O(H2O MASS/FUEL MASS)=0.714
      A(FIRE SPREAD PARAMETER)=0.0110
    OBJECT NUMBER 2 (ID= 2) :
      X-COORD= 2.0800    Y-COORD= 2.8180       HEIGHT= 0.8640
      ANGLE WITH HORIZONTAL=  0.00        ANGLE WITH VERTICAL X-PLANE=  0.00
      THICKNESS= 0.1000                   THERMAL CONDUCTIVITY= 0.0540
      SPECIFIC HEAT= 1900.                DENSITY= 48.00
      EMISSIVITY= 0.98                    CHI(FRACTION OF HEAT RELEASED)= 0.65
      HEAT OF COMBUSTION= 2.870E+07       HEAT OF VAPORIZATION= 2.054E+06
      PYROLIZATION TEMP= 600.0            IGNITION TEMP= 740.0
      AIR/FUEL MASS RATIO= 14.45          STOCHIOMETRIC MASS RATIO=  9.85
      INITIAL MASS= 1.0963                INITIAL RADIUS= 0.0370
      MAXIMUM RADIUS= 0.4657              OBJECT RADIUS= 0.3439
      FCO2(CO2 MASS/FUEL MASS)=1.504      FCO(CO MASS/FUEL MASS)=0.013
      FS(SMOKE MASS/FUEL MASS)=0.241      FH2O(H2O MASS/FUEL MASS)=0.714
      A(FIRE SPREAD PARAMETER)=0.0110
 VENT NUMBER 1:
      WIDTH= 0.7620       HEIGHT= 2.0320          TRANSOM DEPTH= 0.4064
 WALL NUMBER 1:
      THICKNESS= 0.0254                   THERMAL CONDUCTIVITY= 0.1340
      SPECIFIC HEAT= 1062.                DENSITY= 800.0
 PHYSICAL CONSTANTS:
      SPECIFIC HEAT OF AIR= 1004.         RAD MEAN PATH IN FLAME= 0.65
   FOR AIR:
      HEAT TRANSFER COEFF= 10.00          PLUME ENTRAINMENT COEFF= 0.10

   FOR LAYER GASES:
      MAX HEAT TRANSFER COEFF= 50.00      MIN HEAT TRANSFER COEFF= 5.00
   FOR VENTS:
      FLOW COEFF= 0.68


BASIC TIME INCREMENT= 2.00 SECONDS
```

```
 TIME     Z<LEZ(1)     Z<OZZ(2)     ZJLEZ(1)     ZHLZZ(1)     ZYLOZ(1)     TMUZZ(1)    -TMOZZ(1)     ZRFZZ(1)

   0.0   3.0098E+02   3.0000E+02   5.0000E-04   7.8556E-03   2.3146E-01   0.0000E+00   1.1190E-05   3.7316E-02
  10.0   3.0110E+02   3.0000E+02   9.2986E-03   7.2298E-02   2.3142E-01   0.0000E+00   1.4413E-05   4.0744E-02
  20.0   3.0127E+02   3.0001E+02   1.0659E-02   1.5724E-01   2.3137E-01   0.0000E+00   1.9752E-05   4.5432E-02
  30.0   3.0149E+02   3.0001E+02   1.2358E-02   2.4695E-01   2.3130E-01   0.0000E+00   2.7004E-05   5.0509E-02
  40.0   3.0176E+02   3.0001E+02   1.4439E-02   3.4120E-01   2.3121E-01   0.0000E+00   3.6830E-05   5.6325E-02
  50.0   3.0209E+02   3.0002E+02   1.7012E-02   4.3956E-01   2.3111E-01   9.0040E-04   5.0117E-05   6.2534E-02
  60.0   3.0250E+02   3.0002E+02   2.0207E-02   5.3826E-01   2.3098E-01   7.7884E-03   6.8044E-05   6.9594E-02
  70.0   3.0302E+02   3.0006E+02   2.4203E-02   6.3233E-01   2.3081E-01   1.9049E-02   9.2178E-05   7.7271E-02
  80.0   3.0369E+02   3.0009E+02   2.9234E-02   7.1868E-01   2.3060E-01   3.3757E-02   1.2459E-04   8.5732E-02
  90.0   3.0451E+02   3.0013E+02   3.5604E-02   7.9544E-01   2.3034E-01   5.1202E-02   1.6801E-04   9.5054E-02
 100.0   3.0556E+02   3.0017E+02   4.3702E-02   8.6191E-01   2.3001E-01   7.0792E-02   2.2604E-04   1.0532E-01
 110.0   3.0599E+02   3.0024E+02   5.4016E-02   9.1797E-01   2.2958E-01   9.2090E-02   3.0339E-04   1.1661E-01
 120.0   3.0856E+02   3.0032E+02   6.7156E-02   9.6445E-01   2.2904E-01   1.1483E-01   4.0616E-04   1.2903E-01
 130.0   3.1063E+02   3.0044E+02   8.3863E-02   1.0029E+00   2.2835E-01   1.3892E-01   5.4233E-04   1.4267E-01
 140.0   3.1320E+02   3.0059E+02   1.0503E-01   1.0349E+00   2.2747E-01   1.6439E-01   7.2215E-04   1.5765E-01
 150.0   3.1634E+02   3.0081E+02   1.3159E-01   1.0622E+00   2.2637E-01   1.9140E-01   9.5878E-04   1.7407E-01
 160.0   3.2016E+02   3.0109E+02   1.6505E-01   1.0856E+00   2.2500E-01   2.2022E-01   1.2690E-03   1.9207E-01
 170.0   3.2476E+02   3.0149E+02   2.0648E-01   1.1095E+00   2.2330E-01   2.5122E-01   1.6741E-03   2.1175E-01
 180.0   3.3033E+02   3.0203E+02   2.5735E-01   1.1322E+00   2.2120E-01   2.8483E-01   2.2009E-03   2.3327E-01
 190.0   3.3705E+02   3.0276E+02   3.2004E-01   1.1553E+00   2.1864E-01   3.2141E-01   2.8827E-03   2.5674E-01
 200.0   3.4521E+02   3.0376E+02   3.9604E-01   1.1793E+00   2.1553E-01   3.6122E-01   3.7611E-03   2.8232E-01
 210.0   3.5515E+02   3.0415E+02   4.8800E-01   1.2042E+00   2.1177E-01   4.0429E-01   4.8870E-03   3.1015E-01
 220.0   3.6726E+02   3.0597E+02   5.9882E-01   1.2302E+00   2.0724E-01   4.5029E-01   6.3232E-03   3.4037E-01
 230.0   3.8122E+02   3.1001E+02   7.3195E-01   1.2571E+00   2.0181E-01   4.9858E-01   8.1456E-03   3.7313E-01
 240.0   3.9991E+02   3.1663E+02   8.9206E-01   1.2861E+00   1.9523E-01   5.5090E-01   1.0595E-02   4.0873E-01
 250.0   4.2770E+02   3.2762E+02   1.0977E+00   1.3221E+00   1.8665E-01   6.1791E-01   1.4053E-02   4.4840E-01
 260.0   4.5451E+02   3.4656E+02   1.3710E+00   1.3570E+00   1.7527E-01   6.8161E-01   1.8725E-02   4.9295E-01
 270.0   5.1191E+02   3.7820E+02   1.7315E+00   1.3932E+00   1.6026E-01   7.4050E-01   2.5104E-02   5.4314E-01
 280.0   5.7171E+02   4.2995E+02   2.2143E+00   1.4339E+00   1.4002E-01   7.9645E-01   3.3953E-02   6.0002E-01
 290.0   6.4799E+02   5.0812E+02   2.8846E+00   1.4798E+00   1.1168E-01   8.4718E-01   4.6557E-02   6.6514E-01
 300.0   7.4599E+02   6.2404E+02   3.8634E+00   1.5281E+00   8.2484E-02   8.9249E-01   6.5889E-02   7.4122E-01

*** FIRE OF OBJECT 1 LIMITED BY OXYGEN STARVATION AT TIME = 305.25


*** OBJECT  2: STATE  2 -->  5 AT TIME  308.130

*** OBJECT  2 HAS IGNITED


*** FIRE OF OBJECT 2 LIMITED BY OXYGEN STARVATION AT TIME = 308.31

 310.0   8.3538E+02   7.4000E+02   5.6936E+00   1.5624E+00   7.0164E-02   8.9346E-01   9.2705E-02   9.3213E-01
 320.0   8.7381E+02   7.4000E+02   7.6961E+00   1.5793E+00   6.5503E-02   9.0457E-01   1.1775E-01   9.3574E-01
 330.0   8.8959E+02   7.4000E+02   8.8392E+00   1.5850E+00   6.3567E-02   9.0641E-01   1.2594E-01   9.5227E-01
 340.0   9.0208E+02   7.4000E+02   9.0719E+00   1.5849E+00   6.3418E-02   8.9528E-01   1.0470E-01   9.6203E-01
```

```
*** FIRE OF OBJECT  1 CEASES TO BE LIMITED BY OXYGEN STARVATION AT T = 341.38

350.0     7.1556E+02    7.4000E+02    4.8857E+00    1.4204E+00    9.9707E-02    7.3291E-01    5.4927E-03    9.6561E-01

*** FIRE OF OBJECT  2 CEASES TO BE LIMITED BY OXYGEN STARVATION AT T = 349.88

360.0     5.5579E+02    7.4000E+02    3.0533E+00    1.2883E+00    1.3465E-01    6.3628E-01    2.2904E-04    9.6770E-01
370.0     5.3636E+02    7.4000E+02    2.4550E+00    1.2098E+00    1.4368E-01    5.7811E-01    9.1352E-05    9.6770E-01



 TIME     ZKLZZ(1)      ZKOZZ(2)      ZULZZ(1)      ZHLZZ(1)      ZYLOZ(1)      TMUZZ(1)      -TMOZZ(1)     ZR*ZZ(1)

380.0     5.2742E+02    7.4000E+02    2.2939E+00    1.1754E+00    1.4482E-01    5.5509E-01    3.4639E-07    9.6770E-01

*** OBJECT  1: STATE  5 --> 10 AT TIME  380.321 ← Object all burned up.

390.0     5.8708E+02    7.4000E+02    2.1929E+00    1.1018E+00    1.5079E-01    4.5730E-01    0.0000E+00    9.6770E-01
400.0     5.0557E+02    7.4000E+02    1.5149E+00    8.8248E-01    1.7561E-01    2.7052E-01    0.0000E+00    9.6770E-01
410.0     4.9032E+02    7.4000E+02    1.2208E+00    6.9841E-01    1.8520E-01    1.2820E-01    0.0000E+00    9.6770E-01

*** OBJECT  2: STATE  5 --> 10 AT TIME  410.437

420.0     4.8295E+02    7.4000E+02    1.1740E+00    5.5959E-01    1.8514E-01    5.1860E-02    0.0000E+00    9.6770E-01
430.0     4.7140E+02    7.4000E+02    1.1740E+00    4.9407E-01    1.8514E-01    2.2390E-02    0.0000E+00    9.6770E-01
440.0     4.5103E+02    7.4000E+02    1.1741E+00    4.6013E-01    1.8614E-01    1.0658E-02    0.0000E+00    9.6770E-01
450.0     4.5215E+02    7.4000E+02    1.1741E+00    4.4025E-01    1.8614E-01    5.2911E-03    0.0000E+00    9.6770E-01
460.0     4.4447E+02    7.4000E+02    1.1741E+00    4.2743E-01    1.8614E-01    2.5686E-03    0.0000E+00    9.6770E-01
470.0     4.3775E+02    7.4000E+02    1.1741E+00    4.1850E-01    1.8614E-01    1.1121E-03    0.0000E+00    9.6770E-01
480.0     4.3179E+02    7.4000E+02    1.1741E+00    4.1187E-01    1.8614E-01    3.3495E-04    0.0000E+00    9.6770E-01
490.0     4.2545E+02    7.4000E+02    1.1741E+00    4.0660E-01    1.8614E-01    2.2597E-06    0.0000E+00    9.6770E-01
500.0     4.2153E+02    7.4000E+02    1.1741E+00    4.0200E-01    1.8614E-01    0.0000E+00    0.0000E+00    9.6770E-01

ELAPSED CPU TIME:   1 MINUTES, 24.170 SECONDS
```

shrunk because of cooling

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NBS-GCR-81-344	2. Performing Organ. Report No.	3. Publication Date October 1981

4. TITLE AND SUBTITLE

DOCUMENTATION FOR CFC V, THE FIFTH HARVARD COMPUTER FIRE CODE

5. AUTHOR(S)

Henri E. Mitler and Howard W. Emmons

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

Harvard University
Division of Applied Sciences
Cambridge, MA

7. Contract/Grant No.

G7-9011, Am. 10

8. Type of Report & Period Cover

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

National Bureau of Standards
Department of Commerce
Washington, DC 20234

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

This is a complete documentation of the fifth version of the Computer Fire Code (Mark 5). Mark 5 is of course a substantial improvement over Mark 4, which in turn had expanded and generalized Mark 3, etc. Although imposingly thick, this document has been written in a form which may be easily read, as each section begins with a brief outline of that section. The details are then given in the following paragraphs, for the interested reader. The Computer Fire Code permits the calculation of the evolution of a fire in an enclosure with a number of vents, and containing a number of objects (flammable or otherwise). The fire may be of several kinds, and the calculation will proceed for whatever time the user selects. Suggestions for the improvement of the program itself or of this document will be gratefully received. The tape containing the program itself is available at cost.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

Compartment fires; computer programs; fire extinguishment; flashover; ignition; mathematical models; room fires

13. AVAILABILITY

[✓ XX] Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

[✓ XX] Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

187

15. Price

$ 15.50

USCOMM-DC 6043-P